TRUCHAS Physics and Algorithms

The TELLURIDE Team

Version 2.4.0
February 14, 2008

LA-UR-08-0819

LA-UR-08-0819

LA-UR-08-0819

LA-UR-08-0819

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This document contains descriptions of many of the physical models and numerical algorithms in the current release of TRUCHAS. Research and development work is continuing in all areas of physical modelling, numerical modelling, algorithms and implementation. That research and development work is described elsewhere, typically in research papers by the individuals doing the work.

Other documents in this series describe the input varaibles in great detail (TRUCHAS Reference Manual) and give examples of how to use the code (TRUCHAS Users Manual).

Chapter 2

Overview

Truchas is being developed by the Los Alamos National Laboratory (LANL) ASC sponsored TELLURIDE project to provide an effective tool for simulating manufacturing processes. At the same time, we hope to provide a multi-physics computational platform that will be useful for a wider range of simulations that involve materials and/or incompressible flow with multiple fluid interfaces.

The current release of TRUCHAS emphasizes the simulation of metal casting processes as performed at LANL, which has largely determined the models and numerical methods implemented in the code. The models in this release of Truchas include:

- Joule heating
- Phase change
- Thermal Conduction, convection
- Radiative energy exchange
- Binary alloy phase diagram
- Alloy composition tracking
- Incompressible fluid flow
- Interface identification and motion
- Surface tension
- Chemical reactions

- Thermo-mechanical displacement
- Contact and gap formation

This section of the manual will briefly describe each of these models, emphasizing the reasons for the selection of capabilities and limitations as related to the principal applications of TRUCHAS. A few interesting highlights of numerical algorithms are also described here. Later sections of this manual describe the details of each of these models.

2.1 General Issues

Before delving into the specific physical models in TRUCHAS, it is worthwhile to describe a few general characteristics of the code.

2.1.1 Mesh Structures

TRUCHAS follows the classical computational method of solving partial differential equations by subdividing the domain of interest into 'mesh cells' that are treated as independent volumes described by a short list of dependent variables associated with each 'cell'. TRUCHAS simulations may employ two overlapping computational meshes. The electromagnetic solution is always performed on a tetrahedral mesh, while all other solutions use an unstructured grid composed of non-orthogonal hexahedral cells. The hexahedral cells may be used to represent polyhedra of fewer than six faces (tetrahedra, prisms, etc.)

Most TRUCHAS grids are generated by external programs and imported into the code in Exodus II format. TRUCHAS does incorporate a simple orthogonal grid generator that can provide limited control over local refinement in each coordinate direction. The internal grid generator is most useful for creating input for test problems in simple geometry.

The numerical algorithms incorporated in TRUCHAS are designed to work with the range of grids described above. Accuracy is generally best for cells that are orthogonal or nearly so. Therefore, use of a well conditioned grid structure is strongly encouraged.

2.1.2 Time Splitting

TRUCHAS is inherently a transient solution tool. It can be used to establish steady state conditions by following the course of a transient from an initial guess, but this is often an expensive process. With the notable

exception of the integration of the thermal solution with phase change, TRUCHAS advances in time by operator splitting. That is, each physics model is solved sequentially, using the state of all other physics solutions from previous solutions of the corresponding model. The models for phase change, thermal conduction and radiation are solved simultaneously as a coupled set of nonlinear equations, which provides a very stable numerical bedrock for the other physics models.

2.1.3 Homogenization of Material Properties

Mesh cells frequently contain a mixture of materials in TRUCHAS simulations. Each physics model requires values of material properties (density, thermal conductivity, etc.) for such cells. The usual method of evaluating these in TRUCHAS is to treat the cell as a homogeneous mixture of materials, weighting the property by either volume or mass fraction. Variations between mesh cells are explicitly permitted by the algorithms implemented in TRUCHAS. The most significant deviation from this principal of homogenization is the reconstruction of interface geometry in the fluid flow algorithm, as discussed in Section 3.2.1.

2.1.4 Solvers

TRUCHAS makes use of three 'black box' solvers within its numerical algorithms. The UbikSolve package (Appendix C) is used for the solution of coupled sets of linear equations. Coupled non-linear equations are solved by either the Jacobian Free Newton Krylov (JFNK) algorithm or an Accelerate Inexact Newton (AIN) method (Appendix D).

2.1.5 Parallel Paradigm

TRUCHAS employs a rather crude form of parallelism. It is based on a one-time subdivision of the computational mesh into partitions of groups of mesh cells. We use the Chaco program (a product of Sandia National Laboratory) to perform this subdivision. Chaco does not account for the physical characteristics of the various regions of the mesh, only the connectivity of the mesh.

Subsequent to this division of the mesh into partitions, TRUCHAS uses the PGSLIB parallel communications library (a product of Cambridge Power Computing Associates) to provide data from adjacent cells that may reside on different processors. This form of parallel communication generally associates adjacent data with a cell by transferring the data into a local array by way of an explicit call to a PGSLIB routine. This data is then most often referenced in the normal Fortran fashion.

2.1.6 Languages

TRUCHAS is primarily written in FORTRAN 90/95. The source makes substantial use of dynamic memory allocation, the module concept, pointers, interface definitions, array operations, where constructs, and optional arguments. It will fail to compile under earlier versions of FORTRAN, and in fact, it challenges many supposedly capable FORTRAN 90/95 compilers. Check the release notes for each version of the code for a list of tested compilers and environments.

Smaller sections of TRUCHAS are written in C. Python is used extensively in some of our auxiliary programs (particularly the output parser).

2.2 Joule Heating Model

LANL casting operations are mostly performed in vacuum because of the high chemical reactivity of the molten metal alloys of interest. This has led to the use of electromagnetic coils to heat the mold and metal charge by Joule (induction) heating. These coils are run at relatively low frequency (on the order of thousands of hertz) generating fairly long wavelength electromagnetic waves that interact with the mold and charge by inducing electric currents within their walls. These currents produce heat through resistive heating.

The time scales associated with the electromagnetic waves are very short compared to the time needed for the Joule heating to raise the temperature of the mold and charge. The Electromagnetic (EM) model in TRUCHAS is therefore based on a 'snapshot' model of the heatup process. A few cycles of the electromagnetic waves are simulated to produce a periodic solution of the Joule heating within the furnace. The rate of heating is then averaged over the periodic solution to create a spatially varying heating rate that is constant in time. The electrical properties of the furnace will change as its materials heat up, so TRUCHAS provides the ability to automatically recalculate the EM solution based on the changes in properties. Thus, we produce a step-wise constant heating rate that is supplied to the thermal models within the code. Power level changes can also be specified through input to TRUCHAS. Such changes do not require recalculation of the electromagnetic field because the field is a linear function of the driving amplitude.

The EM model uses a mimetic finite-element method to solve Maxwell's equations within a region interior to the heating coils, as described in detail in Section 7.2. The current implementation of this method is limited to tetrahedral grids. Automatic mapping of the cycle averaged Joule heat rate from this EM grid to the thermal mesh (which is usually hexahedral) is included within TRUCHAS. Similarly, mapping of electromagnetic properties from the thermal grid back to the EM tetrahedral mesh is also automatic. The implementation of the EM model also uses quite specific boundary conditions in order to simplify the necessary coding. There are plans to relax these implementation limitations in future versions of TRUCHAS.

2.3 Phase Change and Thermal Models

The alloy charge used in LANL casting operations begins as solid, is melted by the Joule heat described above so that it can be poured, is poured into a mold as a high temperature liquid, solidifies as it cools through contact with the colder walls of the mold, and then undergoes a sequence of allotropic (solid-solid) phase changes as the entire assembly cools to room temperature. Obviously, modeling phase change is important in any simulation of the casting process. These phase changes are also complex. The temperatures at which the phase changes occur depend on the alloy composition, which can vary as the alloy material segregates through the metal charge. The same variables affect the range of temperatures over which the phase change takes place. These relationships are described by the equilibrium phase diagram for the alloy. The properties of the final casting depend critically on the microstructure of the alloy, which in turn depends on such variables as the cooling rate through the phase change and the alloy composition.

Phase change is evidently closely linked to the distribution of heat in the system. TRUCHAS therefore jointly solves the equations for phase change and enthalpy (which is preferred to temperature as an independent variable in problems involving phase change). The set of linked equations is non-linear because thermal properties (specific heat, conductivity) are often functions of temperature, and because of the latent heat that accompanies almost all phase change processes. Within this phase of the solution TRUCHAS approximates each mesh cell as a 'control mass' whose composition is only changed by transformations of phase. This results in a coupled system of equations that includes enthalpy transfer by conduction and radiation, sources that arise from convection and Joule heating, and equilibrium relations between phases that are introduced through a simplified phase diagram.

TRUCHAS does not represent the spatial variation of alloy concentration (coring) within material grains because it calculates cell averaged alloy concentrations for each material. This leads to errors in simulations that involve remelting because it is unable to resolve the time variations that occur as grains melt from the outside inward.

The equation

$$\frac{\partial(\rho h)}{\partial t} = \nabla \cdot (\kappa \nabla T(h)) + Sources \tag{2.1}$$

is a good representation of the overall system of equations to be solved by the coupled algorithm. The spatial derivatives in this equation may be evaluated by any of several discrete derivative approximations as described in Appendix A. The Sources term in this equation represents several effects:

- Physical sources (due to chemical reactions, welding energy deposition, etc.)
- Convective enthalpy source (treated as fixed during this phase of the solution)

- Specified enthalpy fluxes at the boundaries of the domain
- Radiative enthalpy fluxes (discussed in Section 2.4)

TRUCHAS approaches this equation by a backward Euler time difference for which T(h) is included at the new time level, which results in the coupled non-linear equations referred to above. The functional form of T(h) is arbitrary and may be quite complex if the phase diagram is itself complicated. This is dealt with in TRUCHAS by determining T(h) with a cell-by-cell iteration that relates the temperature and phase and alloy composition of the cell to its enthalpy through the phase diagram. The iterative procedure also determines the phase composition of each cell as a byproduct of the evaluation of the cell temperature.

The resulting set of coupled nonlinear equations is solved by either the JFNK or AIN solution procedure, as described in more detail in Appendix D. Both these procedures are iterative in nature, resulting in nested iterations when coupled with the iterative T(h). The advantage of this complex solution procedure is a very stable and robust algorithm. The algorithm will work with arbitrarily large time steps, at least in principal. In practice, we find that it is possible to choose too large a time step, which is manifested by a lack of convergence at some level of the nest. TRUCHAS controls the time step size by a user specified Fourier number, which informs the code of the maximum ratio between the time step selected and the time step that would be stable for an explicit calculation. The stability of this algorithm most often results in time steps that are limited by other numerical algorithms (flow, solid mechanics) in coupled problems. It is generally only in problems that deal solely with the thermal solution that time step difficulties arise.

2.4 Radiative energy exchange

Radiative energy exchange plays two important roles in LANL casting processes.

1. The source from electromagnetic induction is far from uniform during the heat up phase of the process. Conductors heat more rapidly than insulators, and parts of the system are shielded from the electromagnetic waves by parts that are nearer the heating coils. Radiation between system components redistributes this heat, making the temperature more uniform than would otherwise be the case.
2. Heat must ultimately be rejected to the surrounding environment during the solidification and cool down of system components. Transfer of energy to the environment takes place mostly by radiation because the vacuum region inhibits convection and conduction as transfer mechanisms.

TRUCHAS includes both a simple and a complex radiative energy transfer model. Both models are able to extrapolate the cell-centered temperature field (as calculated by T(h)) to the cell faces at which radiation is occurring by implicitly calculating the temperature difference due to the thermal flux passing through the

homogenized cell material. (This extrapolation is always employed by the simple model, but is optional for the complex model.) Both models are also 'gray body' approximations that use a single, spectrum averaged emmisivity. The simple model calculates the enthalpy source term as exchange between the surface temperature and a specified (possibly time varying) temperature. Use of this model usually involves multiple groups of faces radiating to multiple specified temperatures. The simple model always uses an implicit formulation for which the thermal fluxes are updated as part of each iteration of the nonlinear solution described in Section 2.3. The 'Source' terms associated with this model therefore vary during the iterative solution process.

The complex model is based on view factors that are read in by TRUCHAS. (Within the project we use the Sandia National Laboratories Chaparral software to generate these view factors.) View factor based radiation allows the transfer of enthalpy between surfaces within the simulation model, which is crucial to many applications at LANL. The complex model may use either an implicit or explicit time representation, which choice is controlled through input to TRUCHAS. If the implicit representation is chosen an additional iterative solution of coupled equations is introduced because the face-to-face fluxes are related through a set of linear matrix equations.

2.5 Phase Diagrams

The need for phase diagram information is discussed above Section 2.3. This emperical data describes the dependence of each phase transition on alloy composition and temperature. Microsegregation models are necessary to properly model the changes in alloy composition that accompany phase change. Composition is important to the evaluation of any bulk property of the resulting alloy material.

The following of types of phase diagram are available in TRUCHAS:

- Isothermal Phase Change (characteristic of pure materials)
- Non-isothermal Phase Change (an approximation for complex alloys)
- Binary Alloy Phase Change (for alloys of two materials)
- Eutectic Phase Change (for binary alloys with a eutectic)

The isothermal and non-isothermal phase change models do not treat the concentration of any alloy materials. They are useful both for pure materials, and as approximate models of alloys with complex phase diagrams (such as ternaries).

The current TRUCHAS binary alloy phase diagram is quite simplified. The phase diagram assumes that alloy metal concentrations are fairly low so that a linear variation in liquidus and solidus temperature with

concentration is acceptably accurate. An approximate Eutectic phase diagram is offered as an alternative. It too is based on linear variations about zero concentration, but includes a Eutectic point.

Three models of alloy metal microsegregation are available:

- Lever rule (assumes infinite diffusivity in the solid phase)
- Scheil rule (assumes zero diffusivity in the solid phase)
- Clyne and Kurtz model (finite diffusivity in the solid phase)

All three of these models assume that the diffusivity in the liquid phase is effectively infinite. That is, they assume a uniform distribution of solute within the liquid portion of each mesh cell.

2.6 Incompressible fluid flow

TRUCHAS began as a three-dimensional extension of the Ripple [1] code for incompressible fluid flow with sharp interfaces, and it continues to be used by some for similar flow applications. Such flows are clearly also of great interest for many material processing simulations as well. Metal casting and welding are certainly good examples of processes that require accurate simulation of moving interfaces. These interfaces may become topologically complex, forming drops, bubbles, waves, and intersecting interfaces typified by breaking waves and drops impacting on pools of liquid.

Natural convection is important during cooling and solidification in both casting and welding applications. This requires accurate treatment of density changes due to temperature and/or alloy composition, and viscous shear stress, as well as the gravitational force that establishes the pressure gradient that leads to buoyant flow. Both normal and tangential surface tension forces play important roles in welding simulation. Turbulent phenomena greatly enhance the diffusion of momentum and enthalpy and must also be modeled. TRUCHAS includes physical models for all of these phenomena.

A variety of boundary conditions are also necessary for the simulations we carry out with TRUCHAS. The present release includes symmetry conditions, dirichlet pressure and dirichlet velocity boundaries. Although we have a strong interest in periodic boundaries, these have not been implemented because of the difficulties of doing so on unstructured grids.

Some applications also require that the interface model be capable of dealing with vacuum (also often termed 'void' in the flow literature). Even processes that involve metal flow in an atmosphere require the ability to deal with interfaces that separate fluids of vastly different densities. This large ratio of fluid densities (often exceeding 1000:1) places strong demands on the ability of an algorithm to deal with advective processes

that display similar ratios of advected quantities (enthalpy and momentum are both often important). Material processing applications also often involve situations in which slight density differences due to varying temperature or alloy material concentrations lead to buoyant flow in the context of a nominally hydrostatic pressure field. The TRUCHAS flow algorithm has been developed to incorporate accurate transient representation of all of these necessary features.

The algorithm can best be understood as three sequential steps:

- Material advection
- Velocity prediction
- Pressure solve and velocity correction

2.6.1 Material advection step

The basis of the material advection step is the Volume of Fluid (VOF) algorithm which represents the distribution of materials in the problem domain. Material fractions are assigned to each mesh cell of the grid that reflect the instantaneous volumetric fraction of each material in the cell. These volume fractions are accepted in the VOF algorithm as the most complete description of the material distribution. A 'reconstruction' step becomes necessary for the purpose of evaluating a continuum picture of the material interface for the purpose of moving material between mesh cells and determining the interface curvature for evaluating surface tension forces. TRUCHAS uses the Piecewise Linear Interface Calculation (PLIC) [2] to reconstruct material interfaces. There may be more than two materials in some mesh cells of the computational grid, which leads to the necessity of distinguishing among interfaces. TRUCHAS employs the 'onion skin' model that treats multiple interfaces as sequential layers each layer 'on top of' all the previous layers. Use of the onion skin model requires an ordering of materials which TRUCHAS determines from user specified material 'priorities'. The onion skin model reduces the multiple material problem to several simpler problems of determining the interface between two groups of materials. PLIC approaches this simplified problem by first determining a local normal to the interface, and then positioning the plane defined by this normal as to conserve the material volumes on either side within the cell. (Determining this position requires an iterative process because of the unstructured grid used in TRUCHAS.) Determination of the interface normal is performed by a discrete approximation to the gradient of the material fractions.

Movement of the material between cells is based on combining the reconstructed geometry obtained from the PLIC algorithm with the normal component of fluid velocities located on the faces of all mesh cells. (Determination of these is the main objective of the other steps of the flow algorithm.) The discrete divergence of these face velocities must be very nearly zero, or there may be accumulation or depletion of volume within a mesh cell. (Minor accumulations or depletions are dealt with by small adjustments to the volumes entering and leaving a mesh cell.) The advection algorithm first constructs a 'flux volume' which represents the material that moves through a cell face during the course of a time interval for every face velocity that is

pointing out of each cell. (The time interval may be smaller than the overall solution time step, a procedure called 'subcycling the advection'.) The intersection of this flux volume with the internal geometry of the cell (as determined by applying PLIC to the onion skin model) determines a volume of each material that leaves a mesh cell. The same material volumes necessarily enter the cell into which the face velocity points.

The results of this are arrays of material volumes that are transferred from one mesh cell to each of its neighbors in the course of a time step. These material volumes are then used to evaluate new material volume fractions in all mesh cells, and to determine the source terms for all other conservation equations impacted by flow (momentum and enthalpy being the main examples in TRUCHAS). The strength of this approach is that it enforces consistency between the transfer of volume (which is proportional to mass in this incompressible regime) and all other advected quantities. This is essential to the accurate approximation in the presence of large density ratios.

2.6.2 Velocity prediction step

This is the first of the steps used to determine the new-time velocity field by TRUCHAS. It is based on the fluid momentum conservation equation. The velocity prediction step updates the velocity vector located at the centroid of every mesh cell to reflect the influence of:

- Momentum advection
- Viscous stress
- Porous medium drag
- Tangential surface tension force

Additionally, it includes an estimate of the net effect of the pressure gradient, buoyant and normal direction surface tension forces that is based on the state at the beginning of the time step. This net effect we refer to as the dynamic pressure gradient.

This step is performed at the cell centroid because of the difficulty of accurately determining the momentum advection term in the unstructured mesh for problems that may include large density variations. The cell-face based transfers of momentum derived during the advection step provide a natural evaluation of the momentum change associated with the cell centroid.

Momentum advection and tangential surface tension are treated as explicit terms in the time advancement of the cell centered velocity. Porous medium drag is treated implicitly because it is strictly local (affecting only the diagonal matrix elements) and because it helps to stabilize the time advancement. The viscous term is implemented with variable differencing. The time level of this term is determined by user input, and may vary continuously between explicit and implicit. The implicit option permits a larger stable time

step for many problems that are strongly influenced by viscous effects (such as flows that are driven toward a steady condition balancing viscous and other forces). Any degree of implicitness in the viscous term requires solution of a set of coupled linear equations because the cell centered velocities are linked through the discrete approximations to the velocity gradient. Otherwise this step is an explicit advancement in time on a cell-by-cell basis.

2.6.3 Pressure solve and velocity correction step

The velocity field evaluated in the prediction step suffers from two difficulties. First, the time behavior will be unconditionally unstable because the Courant stability limit for incompressible flow is zero. Second, the resulting velocity field does not satisfy the conservation of mass principal, which is equivalent to zero divergence of the velocity (solenoidal field) for incompressible flow. The pressure solve and velocity update step overcomes both of these problems by updating the approximation used in the prediction step for the dynamic pressure gradient to one based on the pressure at the new time level.

We perform this correction first on the velocity at cell faces because we require that the face velocity field be discretely divergence free for the upcoming advection step of the next cycle. This leads to the following internal steps:

- Transfer the velocity field to faces
- Evaluate the change in pressure
- Update face velocities
- Update cell centered velocities

The details of these steps are described in Section 3.2. For present purposes, we simply note that the second of these steps requires formulation and solution of a set of coupled equations based on the requirement of zero discrete divergence of the face velocities.

At the completion of the pressure solve and velocity correction step we have two manifestations of the velocity field, a cell centered velocity field that will serve as the initial field for the prediction step of the next time cycle, and a face field that will be used for the advection step of the next time cycle.

2.7 Surface tension

Surface tension forces play an important role in many welding processes because of the relatively small scale of weld pools (which creates large normal surface tension components through small radius of curvature)

and because of the large thermal gradients (which creates large tangential surface tension forces). TRUCHAS employs the Continuum Surface Tension [3] for both components of surface tension. Surface tension is also significant in many other flow problems that are characterized by small dimensions such as droplet and bubble behavior.

The normal surface tension model calculates the interface curvature at each face of the mesh cell as the divergence of the normal vector to the interface. The evaluation takes place at the cell faces so that the normal component can be included in the 'dynamic pressure gradient' term in both the predictor and projection steps. This approach permits a much more accurate balance between surface tension and pressure gradients, particularly at steady conditions.

The tangential surface tension force is currently evaluated only from the temperature gradient along the interface. (We intend to include the influence of alloy compostion gradient as well.) This gradient is calculated from the usual expansion of the surface tension coefficient as a power series.

Both surface tension models are time explicit. That is, they are based on the interface geometry and temperature field from the previous time step. This explicit treatment leads to stability constraints on the time step size, which are imposed by TRUCHAS at each time step.

The current release of TRUCHAS does not treat surface tension forces well near the mesh boundaries. There is no wall adhesion (contact angle) model in the code, and the calculation of the interface normal direction is faulty in cells that are close to mesh boundaries. We have also observed poor dynamic behavior in problems that involve more than two materials. The next release of TRUCHAS will at least begin to address these known issues.

2.8 Chemical Reactions

Many material processing questions include effects of chemical reactions. Some processes (such as curing) are dominated by chemistry, while in others the chemical reactions are more of an annoyance (corrosion is a good example). The models for chemistry that are included in this version of TRUCHAS are relatively simple. They have been developed to deal with the curing process. TRUCHAS tracks the time evolution of one chemical constituent converting to one other constituent only. This reaction may progress only in one direction. The transient concentrations are determined from initial concentrations, a maximum final concentration, two reaction constants, and assoicated powers of the concentration, as described in Chapter 5. Energy that is released (or absorbed) by the reaction is treated as an explicit source term in the enthalpy conservation equation.

2.9 Solid Mechanics

The response of solid materials to stresses that are induced by thermal transients and gradients are quite important during casting and welding operations. Volume changes due to allotropic phase change can also induce stress and strain in solid bodies. These same effects are also critical to the determination of distortion and residual stresses in the final product of these processes.

An important example of the role played by solid strains in casting operations is the creation and healing of gaps between the pieces from which the mold is assembled, as well as gaps between the mold sections and the solidified metal part. These gaps can have pronounced effects on the path and rate at which heat is removed from the metal, and thereby influence its microstructure as well.

The current TRUCHAS release can calculate displacements, elastic stresses and both elastic and plastic strains for an isotropic material, including stresses and deformations caused by temperature changes and gradients. The volume changes associated with solid state phase changes can also be included in the solution. A variety of traction and displacement boundary conditions can be specified, and this release includes sliding interfaces and contact, restricted to "small" displacements. The model for plastic flow uses a flow stress that depends on strain rate and temperature with no work hardening. Other material behavior such as porosity formation may be added in future versions of the software.

Small displacement sliding interfaces can be specified, with or without a contact algorithm. The interface is defined with gap elements, that are currently constructed by duplicating mesh nodes and element faces on a surface and constructing elements of zero thickness by connecting the coincident faces and nodes in the mesh definition. In the future, elements of finite thickness may be designated as gap elements. The gap elements are currently only used to provide connectivity information to the sliding and contact algorithms, facilitating the parallel implementation.

Sliding interfaces specified without contact (designated "normal constraint" interfaces) allow coincident nodes across an interface to move relative to each other tangential to the surface but not normal to the surface. This is implemented by treating the nodes on the interface as if they are on a free surface and adding constraints that are dependent on the displacement vector of the coincident node on the other side of the interface.

The discretization method used to solve the equilibrium stress equations is based on a node-centered control volume discretization [4, 5]. This algorithm was chosen because it allows the efficient use of the existing mesh data structures and parallel gather-scatter routines. Control volumes are constructed for each node or cell vertex using data from the mesh for fluid flow and heat transfer. Each cell is decomposed into sub-volumes with faces defined by connecting cell centroids, face centroids and edge midpoints as described in Section 6.3. The same section describes the use of finite-element techniques to evaluate the strains required for the equilibrium stress equation. The breakup of the TRUCHAS nominal mesh into sub-cells results in a substantial increase in the degrees of freedom in the solid mechanics solution as compared to the

other physics algorithms that employ this mesh. This leads to large increases in both the dynamic memory requirements and disk space required for simulations that use this model. This may require the use of fewer mesh cells in the basic mesh on computers with limited memory or disk space.

Chapter 3

Fluid Dynamics

The following chapter presents the flow algorithm incorporated into TRUCHAS. The algorithm solves equations of conservation of mass and momentum for any number of immiscible, incompressible fluids, and tracks the interfaces between them. The algorithm has been specially designed to compute accurate solutions for high density ratio flows (e.g. 10^3 to 10^4), as occur when water or molten metal flows within an atmospheric air environment.

The algorithm will also accommodate the presence of void (a zero density, infinitely compressible material), as an alternative to venting a light incompressible fluid. The code has been written to model flow past arbitrarily-placed solid material (e.g. mold), as well as solidifying material, whether this manifests itself as a well-defined interface due to pure material phase change, or as an extended mushy-zone region due to alloy solidification.

Finally, the flow section of TRUCHAS is responsible for evaluating the advection of both enthalpy and solute concentrations. The heat-transfer/phase-change section evaluates conduction and the sources/sinks of solute.

3.1 Physics

3.1.1 Assumptions and Approximations

The assumptions and approximations made by TRUCHAS for fluid flow fall into three categories: basic assumptions, useful approximations, and calculation-specific assumptions.

Basic assumptions are fundamental to the way in which TRUCHAS handles flow. Removal of basic assump-

tions would require completely rewriting the flow algorithm. These include:

The continuum hypothesis: molecular behavior is averaged over small spatial and temporal regions.

Mixture velocity field: TRUCHAS assumes that a single velocity field can be used to describe the flow of all fluids at any point in space. Therefore, boundary layers between fluids must either be ignored, or resolved by the computational mesh. Because boundary layers are often of small dimensions compared to the overall domain of computations, the latter approach is rarely possible.

Incompressibility: the density of any fluid is independent of pressure.

Limited domain: it is assumed that boundary and initial conditions can be defined that limit the domain of the calculation in space and time.

Useful approximations simplify the equations solved by TRUCHAS. Some of these may be relaxed in the future. Their removal would affect only parts of the flow program, and would not require rewriting the entire algorithm. These include:

Property averaging: the properties of mixed fluids are calculated by either volume or mass averaging the individual material properties within each computational cell. Thus, for example, chemical reactions are not treated in the flow solution.

Boussinesq approximation: the density of fluids is permitted to vary with both temperature and solute concentration within TRUCHAS. These density variations only appear as a buoyant force in the flow equations. The usual Boussinesq method is extended to permit polynomial variations, rather than only linear terms.

Constant specific volumes: this is really another statement of the Boussinesq approximation. TRUCHAS ignores the effects of liquids and solids changing their volumes, even in situations involving phase change.

Turbulent flow: viscous stress is calculated from the averaged molecular viscosity, plus a turbulent viscosity based on an algebraic turbulence model.

Newtonian fluids: viscous stress is assumed to be a linear function of the shear rate.

Void regions: void regions are idealizations of regions with zero density. Therefore, the pressure is uniform throughout each void, and the void can be compressed without effect.

Volume tracking: consistent with the Boussinesq approximation, we track the fractional volume of each material throughout the mesh, not its mass. The tracking algorithm can employ partial time steps to reduce flux errors which we refer to as 'sub-cycling' but remains first order in time. Momentum and energy are advected with volume.

Energy transformation: fluid kinetic energy is not converted into internal energy within TRUCHAS. Therefore, internal energy is conserved, not total energy.

Momentum advection: a first order scheme is used to advect momentum. The scheme uses old-time velocity values, but densities that reflect updated material volume fractions.

Spatial and temporal discretization: a semi-implicit time scheme is used in which the pressure gradient is treated implicitly, and all other forces are treated explicitly. This produces a stable, but first order accurate temporal solution. Spatial derivatives on the unstructured grid are evaluated via a first-order least-squares approximation.

Boundary conditions: boundary conditions are generally enforced via the least-squares spatial discretization scheme. This does not enforce rigorous imposition of the requested conditions. The normal component of Dirichlet velocity condition is an exception to this. This component is rigorously imposed, yielding correct volumetric flow rates through the boundary.

Calculation-specific approximations are of two sorts. The first results from the specification of input variables. For example, fluid density for one or more fluids can be specified to be independent of temperature and solute concentration. There are many examples of this type of assumption, which are best understood by examining the `FLOW AND FILLING` chapter of the `User Manual`.

Other calculation-specific assumptions result from the finite resolution of any calculation. If sufficient resolution is used, the impact of these assumptions lessens. They include:

Geometry initialization: the initial volume occupied by materials within mesh cells is evaluated through a Monte Carlo method that results in statistical errors. Material interfaces that coincide with the mesh structure are not subject to this error.

Interface geometry: the location and orientation of fluid-fluid and fluid-solid interfaces within mesh cells are reconstructed from material volume fractions alone. This introduces errors that shrink as resolution improves.

Multiple interfaces: cells that contain more than two materials impose additional uncertainty on the interface geometry.

3.1.2 Equations

The flow algorithm solves equations of conservation of mass and momentum:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0 \tag{3.1}$$

$$\frac{\partial (\rho \mathbf{u})}{\partial t} + \nabla \cdot (\rho \mathbf{u}\mathbf{u}) = -\nabla p + \nabla \cdot \tilde{\tau} + \mathbf{f}_B + \mathbf{f}_S + \mathbf{f}_D \tag{3.2}$$

$\mathbf{u}$ represents velocity, ρ density, p pressure, $\tilde{\tau}$ the shear stress tensor, $\mathbf{f}_B$ any body forces, and $\mathbf{f}_S$ any surface forces. $\mathbf{f}_D$ is a drag force used to describe flow in the presence of a diffuse solid boundary.

As we assume fluids to be Newtonian:

$$\tilde{\tau} = \mu(\nabla \mathbf{u} + \nabla^T \mathbf{u}) \tag{3.3}$$

where μ represents the dynamic viscosity. If the flow is turbulent, then the dependent variables (velocity vector and pressure) in Equation 3.2 are considered ensemble-averaged quantities, i.e., mean quantities after the corresponding fluctuating components (due to turbulence) have been averaged out. The effects of the velocity fluctuations are captured in the turbulent or Reynolds stress tensor, which must be modeled to close the set of equations. Following Boussinesq's hypothesis, we model this turbulent stress as proportional to the velocity gradients like its molecular counterpart, i.e.,

$$\tilde{\tau}_t = \mu_t(\nabla \mathbf{u} + \nabla^T \mathbf{u}) \tag{3.4}$$

The total, effective stress is then given as

$$\tilde{\tau}_{eff} = \tilde{\tau} + \tilde{\tau}_t = \mu_{eff}(\nabla \mathbf{u} + \nabla^T \mathbf{u}) \tag{3.5}$$

where

$$\mu_{eff} = \mu + \mu_t \tag{3.6}$$

We use a simple algebraic model to calculate the turbulent viscosity. The model assumes that the turbulent diffusivity (or kinematic viscosity), ν_t, is proportional to the product of a velocity scale and a length scale:

$$\nu_t \equiv \frac{\mu_t}{\rho} = c_\mu u' \ell \tag{3.7}$$

where u' and ℓ are the turbulent velocity and length scales, respectively, and c_μ is a proportionality constant that has a value of 0.05. The turbulent length scale represents the size of the eddies, which generally varies throughout the domain. We take the approximation of assuming that ℓ is constant in a problem, with a value specified via code input (e.g., half the dimension of a representative region in a mold cavity). The velocity scale is related to the local turbulent kinetic energy per unit mass, k, which is modeled as a fraction of the mean kinetic energy because in mold filling problems, turbulence is largely derived from the mean flow:

$$u' = \sqrt{k} \tag{3.8}$$

$$k = f \cdot \frac{1}{2} \cdot \mathbf{u} \cdot \mathbf{u} \tag{3.9}$$

Though not true in general, we assume the fraction f to be constant at 0.1 throughout the problem. This default value of the ratio between turbulent and mean kinetic energies may be changed via input.

The only surface force $\mathbf{f}_S$ that we consider is surface tension, which is modeled as a volumetric force acting on fluid in the vicinity of surfaces S:

$$\mathbf{f}_S = (\sigma\kappa\hat{n}_S + \nabla_S\sigma)\delta_S \tag{3.10}$$

σ is the surface tension coefficient, κ the total curvature of an interface, $\hat{n}_S$ a unit normal to S, ∇_S is the surface gradient and δ_S the Dirac delta function.

Although individual fluids are incompressible, we consider multiple immiscible fluids (of different densities) within a domain, and so retain ρ within the bracketed terms on the LHS of Equation 3.2. However, as the density of any fluid particle remains constant,

$$\frac{D\rho}{Dt} = 0 \tag{3.11}$$

and so:

$$\nabla \cdot \mathbf{u} = 0 \tag{3.12}$$

Equation 3.2, then, is an Eulerian expression of conservation of fluid momentum, subject to the incompressibility constraint embodied by Equation 3.12. Equation 3.1 describes the transport of density, or put more simply, the transport of different fluids within the domain.

3.1.3 Initial & Boundary Conditions

A variety of initial and boundary conditions are currently implemented in TRUCHAS, and more will be added in the near future.

Every computational cell in the mesh is automatically provided with initial conditions as specified in the `BODY` namelist inputs. Each such namelist describes a region of space (as described in the `BODY` section of the `Reference Manual`) and provides initial conditions within that region. (Alternatively, there are user modifiable `OVERWRITE` routines that can be used to impose more general initial conditions.) The initial conditions used by the flow algorithm are: material composition, temperature, velocity and solute concentration. TRUCHAS evaluates cell values of the fluid velocity, enthalpy, density, viscosity and surface tension from the values specified by all `BODY` namelists.

Because the geometric regions specified in the namelist may not coincide with the computational grid, TRUCHAS uses a Monte Carlo algorithm to estimate cell values. A sequence of test points are generated for each mesh cell in a pseudo-random fashion, and each point is geometrically assigned to a `BODY` namelist (which may be the `BACKGROUND`). The cell values are estimated by the assuming that the volumetric fraction of the cell occupied by each `BODY` is equal to the fraction of the random points that fall within the `BODY` geometry.

Boundary conditions are evaluated by TRUCHAS at every external face of the mesh. In some cases, boundary conditions can also be imposed in the mesh interior. Conditions are specified by the user through BC namelists. Each namelist specifies a region of space, a boundary condition type and a boundary condition variable, and may specify value(s) for the variable. TRUCHAS determines the cell faces that fall within the spatial extent of the namelist and creates data structures that are used to impose the desired conditions within the solution.

In the case of fluid flow, the currently available boundary variables are fluid velocity and pressure. At solid surfaces and mesh boundaries, no-slip and free-slip velocity conditions can be specified. At mesh boundaries, Dirichlet values may be specified for either pressure or velocity.

Future additions to TRUCHAS will implement a wider variety of flow boundaries, including Neumann, periodic, symmetric and hydrostatic conditions.

3.1.4 Interaction With Other Physics

The principal interactions of the flow solution are with the heat transfer / phase change solution. The flow solution evaluates the redistribution of materials, enthalpy and species concentration due to advection effects. These changes are passed on to the heat transfer / phase change solution. The heat transfer / phase change solution evaluates material changes due to phase change, and temperature variations due to all phenomena. These in turn affect the flow solution through the material properties of the fluids.

Additional interactions will be implemented in the near future. These include Lorentz forces on the fluid due to electromagnetic effects, volumetric changes resulting from density variations that occur with phase change, and displacement volumetric changes evaluated by the themomechanical solution.

3.1.5 Material Properties

The following are material properties associated with the solution of a flow field:

Density: (required) for each material, a density ρ must be specified, that may vary as a function of temperature or temperature plus solute concentration.

Viscosity: (optional) if viscous effects are to be modeled, then a dynamic viscosity μ must be specified for each material, that can vary as a function of temperature or temperature plus solute concentration.

Surface tension: (optional) if surface tension effects are to be modeled, then a surface tension coefficient σ must be specified for each pair of fluids that may come in contact. σ can be specified as a function of temperature.

3.2 Algorithms

The solution of the equations presented in Section 3.1.2 proceeds in a sequence of steps.

The distribution of materials throughout the

mesh is considered first by solving several finite volume analogs of Equation 3.1. TRUCHAS employs the multidimensional PLIC (piecewise linear interface calculation [2]) method to evaluate the volume fraction of every fluid material in every mesh cell. (Note that the current coupling with the phase change model is explicit, that is, the conversion of liquid materials to/from the corresponding solid materials takes place in the enthalpy solution after the flow solution is complete.) Section 3.2.1 describes the volume fraction solution in more detail.

This first step produces not only new values of the volume fractions, but also volume transfers between pairs of cells sharing a face. These volume transfers are used to evaluate the transport of all other quantities to ensure consistency among the various transport equations. Therefore, enthalpy changes and species changes are also derived from the volume transfer data.

Next, the new volume fractions are used to evaluate fluid properties in every mesh cell. The fluid density and viscosity are evaluated as volume averages of the fluid materials within the cell. (The value of each material's density and viscosity can also depend on the material temperature and species concentrations, as discussed in Section 3.2.4)

Once TRUCHAS has determined fluid properties for every mesh cell, it proceeds to the evaluation of new velocity and pressure fields. This evaluation itself is divided into 4 steps:

1. Approximate the new-time cell-centered velocities by a forward Euler step in time, using known values to evaluate momentum advection and the surface tension. These known values are a combination of previous time step values (velocity, temperature, and species concentrations) and material volume fractions and material transfers from the volume tracking step. This "predictor" step also incorporates explicit approximations to the body force and the pressure gradient. These approximations are updated in step 3. Viscous and drag forces are treated more implicitly during this step to enhance stability. Drag forces (which are diagonal terms in the equations) are proportional to the predicted velocity components. Viscous forces are averaged between the previous time step solution and the predicted components (based on a user specified parameter). This imposes a requirement for solving a linear system of equations in most calculations.
2. Evaluate velocities on the cell faces from these cell centered values, and then apply body force accelerations. A "Rhie-Chow" [6] correction is applied during this step to eliminate "checkerboard" spatial oscillations.
3. Project the face velocity field onto a solenoidal field by solving for the change in the pressure field

that eliminates the velocity divergence in every mesh cell.

4. Update the cell-centered velocity field by averaging the face-centered gradients of the change in the pressure field that were evaluated in step 3.

Completion of these steps results in velocity and pressure fields that are fully updated using the forward Euler time step.

The subsections below provide additional details on each of the above steps.

3.2.1 Interface Kinematics

Computational simulations of the mold filling portion of a casting process demand an accurate and robust algorithm for tracking the molten metal free surface. Many candidate algorithms are available today, and many more are likely to be devised; see [7,8] for reviews. The requirements list we impose upon an interface tracking algorithm optimal for mold filling simulations is long and formidable. We seek an algorithm that:

- is globally *and* locally mass conservative;
- maintains at least second order accuracy in time and space;
- maintains compact interface discontinuity width;
- is topologically robust;
- is amenable to three-dimensions;
- is amenable to general unstructured meshes;
- can accommodate additional interfacial physics models;
- can track interfaces bounding more than two materials;
- is computationally efficient;
- can be implemented by novices; and
- can be readily maintained, improved, and extended.

We have expended considerable time and effort in quantitatively comparing most popular candidate algorithms in 2-D [9]. While equivalent 3-D comparison studies have yet been undertaken, we currently conclude that a clear interface tracking algorithm "winner" is not apparent at this time. Each algorithm has readily

identifiable strengths and weaknesses, which, if understood and quantified, could result in a hybrid, unified algorithm possessing the strengths of many different algorithms.

To date we have embraced volume tracking algorithms, where we have found them useful on 2-D structured [10] and unstructured [11] meshes as well as 3-D structured meshes [12]. We have devised and implemented volume tracking algorithms which reconstruct piecewise linear (planar) fluid interfaces from discrete fluid volume data. If the piecewise linear interface reconstruction geometry is linearity-preserving, i.e., reconstructs planar interfaces exactly, then we declare the algorithm to be spatially second order. By detailing the algorithm's extension to 3-D unstructured meshes in the following, the first seven requirements previously listed have at least been addressed (although not adequately); work remains before the remaining four requirements can be addressed, and focused analysis is needed on all requirements before victory can be declared. Volume tracking has been our choice to date because other algorithms have fallen short in satisfying some of the more important requirements such as topological robustness and the maintenance of local conservation and compact interface width.

3.2.1.1 Representing the Interface with Volume Fractions.

We solve Equation 3.1 for ρ^{n+1} using $\mathbf{u}_f^n$. We begin by defining a volume fraction f_k as the fraction of a cell volume V occupied by fluid k:

$$f_k = V_k/V \tag{3.13}$$

A cell density is related to the volume fractions via:

$$\rho = \sum f_k \rho_k \tag{3.14}$$

and Equation 3.1 may then be written as:

$$\frac{\partial(f_k\rho_k)}{\partial t} + \nabla \cdot (f_k \rho_k \mathbf{u}) = 0 \tag{3.15}$$

Since each ρ_k is constant, we obtain an evolution equation for the f_k:

$$\frac{\partial f_k}{\partial t} + \nabla \cdot (f_k \mathbf{u}) = 0 \tag{3.16}$$

Our volume tracking algorithm seeks discrete numerical solutions to

$$\frac{\partial f_k}{\partial t} + \mathbf{u} \cdot \nabla f_k = -f_k \nabla \cdot \mathbf{u} = 0\,, \tag{3.17}$$

where $\mathbf{u}$ is the flow velocity and f_k is the volume fraction of material k. (The final equality only holds in fluid regions that include no "void" material as described below, because the velocity field is not required to

be solenoidal in the vicinity of void.) Here we invoke a one-field approximation, as derived in [8]. Since f_k delineates the presence (or absence) of each fluid, f_k serves as a Heaviside function H for each material k. Equation 3.17 is therefore an evolution equation for the location of each fluid, with the volume fractions f_k discretely approximating H. The volume fractions f_k are bounded by $0 \leq f_k \leq 1$, where

$$f_k = \begin{cases} 1, & \text{inside fluid } k\,; \\ > 0, < 1, & \text{at the fluid } k \text{ interface}\,; \\ 0, & \text{outside fluid } k\,. \end{cases} \tag{3.18}$$

Since fluid volumes are volume-filling, volume fractions must sum to unity, $\sum_k f_k = 1$, throughout the domain. In seeking solutions to Equation 3.17, fluid *volumes* are marched forward in time as solutions to the volume integral of Equation 3.17 [10].

3.2.1.2 An Overview of the Volume Tracking Algorithm.

We utilize a multidimensional PLIC (piecewise linear interface calculation [2]) volume tracking algorithm for unstructured meshes [13] to solve Equation 3.16 for f_k^{n+1}. The algorithm consists of two steps: a planar reconstruction of fluid-fluid interfaces within a cell, corresponding exactly to the f_k^n and to estimates of the orientations of the interfaces (evaluated as gradients of the f_k^n); and then a geometric calculation of volume fluxes of different materials across cell faces. Multiplying these fluxes by Δt provides the volume of each material crossing every face, which is used to update the volume fraction in every cell, and later for transport of other quantities. It is crucial that the transport of each quantity be evaluated from the volume change of each material and not from the average quantity and total volume change.

TRUCHAS allows the use of multiple passes through the volume tracking algorithm within a time step. Sub-cycling improves the accuracy of the solution, and permits transfer of material through more than a single cell during a time step. This can be quite important in regions where the interface travels at an angle to mesh cell faces. Refer to the `CAVEATS` chapter of the `User Manual` for a discussion about the limitations of sub-cycling.

Key differentiating aspects of a given volume tracking algorithm include the temporal integration scheme and the accuracy with which fluid truncation volumes at control volume faces are estimated. Truncation volumes follow from a required reconstructed interface geometry assumption. For this work, we fit the fluid volume data to a reconstructed interface whose geometry is piecewise linear (planar), given by the equation

$$\hat{\mathbf{n}} \cdot \mathbf{x} - \rho = 0\,, \tag{3.19}$$

where $\mathbf{x}$ is a point on the plane and ρ is the plane constant. This approximation is a good one if the radius of curvature of the interface is at least two to three times the characteristic mesh spacing.

We now summarize our volume tracking algorithm template:

1. Estimate the interface topology from discrete f_k data. For piecewise linear schemes, this requires an estimation for the interface normal $\hat{\mathbf{n}}$.

2. Reconstruct the interface in each cell by locating the interface surface within the cell in a volume conservative manner. For piecewise linear schemes, this requires finding the plane constant ρ in Equation 3.19.

3. Define the flux volume boundaries at each control volume face.

4. Compute the fluid volume truncated by the interface surface within each flux volume (V_{tr}).

Various methods for accurate estimation of the interface normal $\hat{\mathbf{n}}$ can be found in [12, 14]; in the following we detail how V_{tr} can be computed exactly within volumes bounded by logical hexahedra typical of most unstructured meshes.

3.2.1.3 Estimating the Interface Normal.

The algorithms used for estimating the interface normal $\hat{\mathbf{n}}$ from discrete material k volume fraction f_k data are discussed in Section 3.2.3.

3.2.1.4 Locating the Interface.

Let the truncation volume V_{tr} be the volume of the portion of the interior of the hexahedron behind the interface plane p. We solve the following two problems iteratively until the truncation volume V_{tr} converges to V_k

- the *direct* problem: given the cell, ρ, and $\hat{\mathbf{n}}$, find V_{tr}; and
- the *inverse* problem: given the cell, $\hat{\mathbf{n}}$, and V_{tr}, find ρ.

The truncation volume is given by

$$\begin{aligned} V_{\mathrm{tr}} &= \int_{\mathrm{v}} 1 \, d\tau = \frac{1}{3} \int_{\mathrm{v}} \nabla \cdot (\mathbf{x} - \hat{\mathbf{n}}\rho) \; d\tau(\mathbf{x}) \\ &= \frac{1}{3} \Big[\sum_{f=1}^{6} \int_{\mathrm{tr}} (\mathbf{x} - \hat{\mathbf{n}}\rho) \cdot d\mathbf{S}_f(\mathbf{x}) + \int_{\mathrm{tr}} (\mathbf{x} - \hat{\mathbf{n}}\rho) \cdot d\mathbf{S}_p(\mathbf{x}) \Big] , \end{aligned} \tag{3.20}$$

where $d\tau$ is an element of volume, $d\mathbf{S}_f$ is a vector element of surface on cell face f, and $d\mathbf{S}_p$ is a vector element of surface on the truncating plane. The surface integrations above are confined to the portions of the surfaces behind the plane (tr), and the $d\mathbf{S}_f$ elements point along the outward normals on each S_f. In Equation 3.20, since $d\mathbf{S}_p = \hat{\mathbf{n}}|\mathbf{S}_p|$, then $(\mathbf{x} - \hat{\mathbf{n}}\rho) \cdot \hat{\mathbf{n}} = 0$, hence

$$V_{\rm tr} = \sum_{f=1}^{6} V_f \,, \quad \text{where} \quad 3V_f = \int_{\rm tr} (\mathbf{x} - \hat{\mathbf{n}}\rho) \cdot d\mathbf{S}_f(\mathbf{x}) \,. \tag{3.21}$$

Hence, to compute $V_{\rm tr}$, we consider each truncated face f separately.

In performing the surface integrals above, we first define the properties of each control volume, which is a computational cell characterized as a *logical cube* having eight vertices, twelve edges, and six faces; vertex positions arbitrary; edges that are straight lines; and faces that are ruled surfaces. Note that this definition easily allows the cell *in physical space* to be a tetrahedron, prism, pyramid, or hexahedron, since any of the eight vertex physical coordinates can coincide.

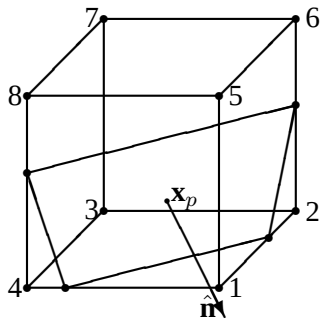


Figure 3.1: A logical cube truncated by a plane whose constant ρ in Equation 3.19 is $\hat{\mathbf{n}} \cdot \mathbf{x}_p$, where $\mathbf{x}_p$ is a point on the plane.

Before defining a ruled surface, consider the following definitions for the four-vertex cell face shown in Figure 3.2. Let the four vertices of a face be labeled $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}_3$, and $\mathbf{x}_4$, ordered counterclockwise, with the outside of the face above the plane of the paper, as shown in Figure 3.2.

The lines $(\mathbf{x}_1, \mathbf{x}_3)$ and $(\mathbf{x}_2, \mathbf{x}_4)$ are therefore diagonal pairs. The successor vertex, $\mathbf{x}'_i$, is defined as the vertex next to and ahead of $\mathbf{x}_i$ in the counterclockwise rotation, i.e., $\mathbf{x}'_1 = \mathbf{x}_2$, $\mathbf{x}'_2 = \mathbf{x}_3$, $\mathbf{x}'_3 = \mathbf{x}_4$, and $\mathbf{x}'_4 = \mathbf{x}_1$. Similarly, the double-successor vertex, $\mathbf{x}''_i$, is the diagonal partner vertex, and the triple-successor vertex, $\mathbf{x}'''_i$, is the predecessor vertex, i.e., $\mathbf{x}'''_1 = \mathbf{x}_4$.

Several vectors and scalars associated with a ruled surface can be defined. First, a deviation vector $\mathbf{B}$ is given by

$$\mathbf{B} = \mathbf{x}_1 - \mathbf{x}_2 + \mathbf{x}_3 - \mathbf{x}_4 \,, \tag{3.22}$$

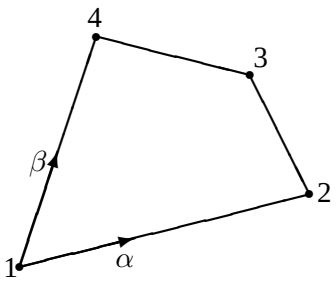


Figure 3.2: A ruled surface is defined by four (in general nonplanar) points connected by straight lines. the surface is parameterized by α and β.

hence $\mathbf{B} = 0$ only if the face is a parallelogram. When $|\mathbf{B}| \neq 0$, then $|\mathbf{B}|$ measures the deviation of the ruled surface from the parallelogram configuration. Next, the vector $\mathbf{k}$, given by

$$\mathbf{k} = (\mathbf{x}_3 - \mathbf{x}_1) \times (\mathbf{x}_4 - \mathbf{x}_2)\,, \tag{3.23}$$

possesses a magnitude which is twice the face area if the face $\mathbf{x}_i$ lie in a plane. The area of the ruled surface in Figure 3.2 is also $|\mathbf{k}|/2$, *independent* of whether the four $\mathbf{x}_i$ lie in a plane. A volume $v_{\rm tet}$, given by

$$v_{\rm tet} = (\mathbf{x}_1 - \mathbf{x}_2) \times (\mathbf{x}_2 - \mathbf{x}_3) \cdot (\mathbf{x}_3 - \mathbf{x}_4)\,, \tag{3.24}$$

is six times the volume of the tetrahedron formed by the four $\mathbf{x}_i$ in Figure 3.2. This volume is zero only if the four $\mathbf{x}_i$ lie in a plane. The cross-vectors $\mathbf{X}_i$ are defined by

$$\mathbf{X}_i = (\mathbf{x}_i''' - \mathbf{x}_i) \times (\mathbf{x}_i - \mathbf{x}_i')\,, \tag{3.25}$$

e.g., $\mathbf{X}_1 = (\mathbf{x}_4 - \mathbf{x}_1) \times (\mathbf{x}_1 - \mathbf{x}_2)$. The cross-vector magnitude is twice the area of the surface bounded the three vertices in its definition. We also define the signs ϵ_i, $1 \leq i \leq 4$ as $\epsilon_1 = \epsilon_3 = +1$ and $\epsilon_2 = \epsilon_4 = -1$. Given the definitions above, we see that $2v_{\rm tet} = \mathbf{B} \cdot \mathbf{k}$, and, if $|\mathbf{B}| = 0$, then $\mathbf{X}_1 = \mathbf{X}_2 = \mathbf{X}_3 = \mathbf{X}_4$.

Let α and β in Figure 3.2 be parametric variables with ranges $0 \leq \alpha \leq 1$, $0 \leq \beta \leq 1$. Then $(1-\alpha)\mathbf{x}_1 - \alpha\mathbf{x}_2$ and $(1-\alpha)\mathbf{x}_4 - \alpha\mathbf{x}_3$ are points on the lines $(\mathbf{x}_1, \mathbf{x}_2)$ and $(\mathbf{x}_4, \mathbf{x}_3)$, respectively. Given this relationship, one can write an expression for any point $\mathbf{x}$ on a ruled surface as

$$\mathbf{x} = \mathbf{x}_1 + \alpha(\mathbf{x}_2 - \mathbf{x}_1) + \beta(\mathbf{x}_4 - \mathbf{x}_1) + \alpha\beta\mathbf{B}\,. \tag{3.26}$$

If the face is planar, then $\mathbf{x}$ is a point inside the quadrilateral, but, more generally, $\mathbf{x}(\alpha, \beta)$ given by Equation 3.26 is a 2-D surface segment of 3-D space whose borders are the straight lines $(\mathbf{x}_1, \mathbf{x}_2)$, $(\mathbf{x}_2, \mathbf{x}_3)$, $(\mathbf{x}_3, \mathbf{x}_4)$, and $(\mathbf{x}_4, \mathbf{x}_1)$. Through any point (α_0, β_0) on this surface, there are two straight lines, namely $\mathbf{x}(\alpha_0, \beta)$, $0 \leq \beta \leq 1$, and $\mathbf{x}(\alpha, \beta_0)$, $0 \leq \alpha \leq 1$, through it which lie entirely on the surface. Thus it is

called a *ruled* surface. The ruled surface can, under suitable translation and rotation of coordinates, also be considered a parabolic hyperboloid whose surface area is the minimum area that can be passed through its four straight lines.

The ruled surface element **dS** is given by

$$\mathbf{dS} = \left[\mathbf{X}_1 + \alpha(\mathbf{X}_3 - \mathbf{X}_4) + \beta(\mathbf{X}_3 - \mathbf{X}_2)\right] d\alpha d\beta \,. \tag{3.27}$$

Integrating **dS** over the ruled surface, on obtains:

$$\int_0^1 d\alpha \int_0^1 d\beta \left[\mathbf{X}_1 + \alpha(\mathbf{X}_3 - \mathbf{X}_4) + \beta(\mathbf{X}_3 - \mathbf{X}_2)\right] = \frac{1}{2}\mathbf{k} \tag{3.28}$$

The trace of the ruled surface on the truncating plane given by Equation 3.19 is a hyperbola on that plane, and the trace of the truncating plane on the ruled surface is a hyperbola on the (α, β) surface.

Let $\mu_i = \hat{\mathbf{n}} \cdot \mathbf{x}_i$, where i denotes any of the four vertices on the ruled surface. Further, define $\rho_i = \min(\rho, \mu_i)$, hence $\rho - \rho_i = 0$ if $\mathbf{x}_i$ is in front of the plane, otherwise $\rho - \rho_i = \mu_i$. Let the $\mathbf{x}_i$ be relabeled $\mathbf{x}_a$, $\mathbf{x}_b$, $\mathbf{x}_c$, and $\mathbf{x}_d$ and set $\mu_a = \hat{\mathbf{n}} \cdot \mathbf{x}_a$, $\mu_b = \hat{\mathbf{n}} \cdot \mathbf{x}_b$, $\mu_c = \hat{\mathbf{n}} \cdot \mathbf{x}_c$, and $\mu_d = \hat{\mathbf{n}} \cdot \mathbf{x}_d$ according to the convention that: $\mu_a \leq \mu_b \leq \mu_c \leq \mu_d$. Given this convention, then, as the plane moves with increasing ρ from $\rho = -\infty$ to $\rho = \infty$, $\mathbf{x}_a$ is the first vertex passed, then $\mathbf{x}_b$, $\mathbf{x}_c$, and $\mathbf{x}_d$ are successively passed. Successor vertices will also be denoted a', a'', so that $a'' = a + 2$. With this notation, all possible plane/ruled surface truncation alternatives divide into six unique cases for V_f:

Case 0: $\rho \leq$ all μ_i, then $V_f = 0$;

Case 1: $\mu_a < \rho \leq \mu_b$, then V_f has only one truncated corner;

Case 2: $\mu_a \leq \mu_b < \rho < \mu_c \leq \mu_d$ and $b \neq a + 2$;

Case 3: $\mu_c \leq \rho < \mu_d$, then only one corner has not been truncated;

Case 4: $\mu_d \leq \rho$ ($V_f = V_{f\mathrm{tot}}$);

Case 5: $\mu_b < \rho < \mu_c$ and $b = a + 2$, which *can* occur for "bow-tied" surfaces.

Given the integral, K_{nm}, defined as

$$K_{nm} = \int_{\mathrm{tr}} \alpha^n \beta^m \, d\alpha \, d\beta \,, \tag{3.29}$$

then V_f can be expressed as

$$3V_f = (\mathbf{x}_1 - \hat{\mathbf{n}}\rho)\cdot\big[\mathbf{X}_1 K_{00} + (\mathbf{X}_3 - \mathbf{X}_4)K_{10} + (\mathbf{X}_3 - \mathbf{X}_2)K_{01}\big] - v_{\rm tet}K_{11}\,. \tag{3.30}$$

Upon analytically performing the K integrals, a general expression for V_f follows:

$$V_f = \frac{1}{6}\sum_i \epsilon_i Y_i \frac{(\rho-\rho_i)^2}{\lambda_i} + \frac{v_{\rm tet}}{2}\sum_i \epsilon_i\big[J_1(w_i) - 2J_2(w_i) + J_3(w_i)\big]\frac{(\rho-\rho_i)^4}{\lambda_i^2}\,, \tag{3.31}$$

where

$$\begin{aligned} Y_i &= (\mathbf{x}_i''' - \hat{\mathbf{n}}\rho)\cdot(\mathbf{x}_i - \hat{\mathbf{n}}\rho)\times(\mathbf{x}_i' - \hat{\mathbf{n}}\rho)\,; \\ \lambda_i &= \epsilon_i(\mu_i - \mu_i')(\mu_i - \mu_i''')\,; \\ w_i &= \frac{\nu}{\lambda_i}(\rho-\rho_i)\,, \quad \text{where} \quad \nu = \hat{\mathbf{n}}\cdot\mathbf{B}\,; \end{aligned} \tag{3.32}$$

and

$$J_1(w) - 2J_2(w) + J_3(w) = 2\sum_{n=0}^{\infty}\frac{(-w)^n}{(n+2)(n+3)(n+4)}\,. \tag{3.33}$$

If w is large, i.e., $w > 10^{-2}$, we compute J_0 and J_1, J_2, J_3 successively by recursion. But if w is small, i.e., $w \le 10^{-2}$, then compute according to the power series in Equation 3.33.

The expression for V_f in Equation 3.31 is quite general, being appropriate for cases 1, 3, and 5 mentioned previously. Cases 0 and 4 are trivial, but Equation 3.31 may break down for case 2 because one of the λ's in the denominator can vanish. In this case, the terms in Equation 3.31 need to be rearranged.

It is useful to define $V_{f\mathrm{tot}}$, which is our case 4, where all four vertices are behind the interface plane, i.e., all $\mu_i \leq \rho$. Using Equation 3.30, and integrating over the entire ruled surface, which yields $K_{00} = 1$, $K_{01} = K_{10} = 1/2$ and $K_{11} = 1/4$, gives:

$$V_{f\mathrm{tot}} = \frac{1}{2}\left(\mathbf{x}_{\mathrm{cm}} - \hat{\mathbf{n}}\rho\right) \cdot \mathbf{k}\,, \qquad \mathbf{x}_{\mathrm{cm}} = \frac{\mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3 + \mathbf{x}_4}{4}\,. \tag{3.34}$$

3.2.1.5 Representing Interfaces Bounding More Than Two Fluids.

Almost two decades ago a simple "onion skin" [15] model was proposed as an means for volume tracking algorithms to represent multiple (> 2) immiscible fluids residing within a single cell. In this instance, the algorithm must accommodate for the reconstruction of more than one interface bounding different sets of fluids within a given cell. Given an adequately-resolved computational domain tasked to represent all interfaces bounding each of n immiscible fluids present in the domain, a surprisingly frequent occurrence is indeed this situation, namely more than two distinct fluids entering a single cell.

The presence of any cells containing more than two fluids is arguably a good criterion for "under resolution", i.e., further mesh refinement of the domain is warranted before the simulation should be continued, because an adequate interface representation could not possibly result otherwise. Consider, however, a simple three-fluid simulation where all three fluids are initially separated from one of the other fluids (i.e., interfaces bounding fluids one-two and two-three are initially present). As the simulation progresses, however, and the fluids begin to move in response to applied dynamics, it is certainly plausible that a fluid one-three interface might arise and even come in contact with a one-two interface and a two-three interface. The prototypical example is of course a triple point between these three fluids. For a complex topology simulation, it is also likely that this triple point does not align with cell boundaries, hence a volume tracking algorithm must therefore be able to accommodate this occurrence (> 2 fluids in a cell) in some way.

Two basic algorithmic approaches can be taken: (1) abandon the immiscibility assumptions (that fluids are separated by distinct interfaces), hence do not attempt to further volume track the interfaces, but instead transition over to a "homogeneous mix" model; or (2) maintain the immiscibility assumptions and continue to volume track the interfaces by breaking down the multiple interface (more than one per cell) problem into a series of two-fluid, one-interface problems. For approach (1), a homogeneous mix of two or more fluids within a given cell refers to the situation where the fluids are intimately mixed with a characteristic interfacial length scale (e.g., local radius of curvature) that is sufficiently smaller than a length scale resolvable by the cell size. Such homogeneous mix will (and does) occur, and the formulation of plausible models meant to capture this phenomenon is quite possible, but such a model does not yet exist in TRUCHAS. Instead, approach (2) is currently supported, which is virtually identical to the 1982 approach published by David Youngs in his classic volume tracking article [15]. The Youngs' onion skin algorithm discussed below has undergone evolutionary developments by Mosso and others since 1982, but these developments have not yet been (and should be) incorporated in the current algorithm. The basic Youngs approach is outlined below, with some concluding remarks on how (and where) the algorithm can be problematic and further improved, including the transition to a homogeneous mix model.

The basic premise is that a volume tracking representation is always reducible to a two-fluid, one-interface problem given by a distribution of volume fraction data $f_{\rm ag}$, where $f_{\rm ag}$ is the *agglomerated* volume fraction, constructed by summing volume fractions in a specific *priority order*:

$$f_{\rm ag(i)} = \sum_{p=1}^{i} f_{k(p)} \,, \tag{3.35}$$

where i is the interface currently being constructed (1, 2, etc.), p the priority number, and $k(p)$ is a mapping of priority number to fluid identification (typically a number). Figure 3.3 displays the generic problem (in 2-D) for defining and locating a piecewise linear interface in a cell containing two fluids, namely where $f_{\rm ag(i)}$

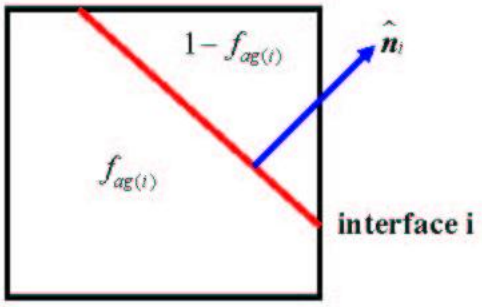


Figure 3.3: A volume tracking representation is always reducible to a two-fluid, one-interface problem given by a distribution of *agglomerated* volume fraction data $f_{\rm ag}$ separated by an interface i having a unit normal $\hat{\mathbf{n}}(i) = \nabla f_{\rm ag(i)}$.

is the volume fraction data on one side of interface i having unit normal $\hat{\mathbf{n}}(i)$ ($= \nabla f_{\rm ag(i)}$) and $1 - f_{\rm ag(i)}$ is the data on the opposite side. For an explicit example, consider the four fluids distributed in one cell as shown in Figure 3.4, $k(p) = 1, 2, 3, 4$ for $p = 1, 2, 3, 4$, respectively, so for the first interface, $f_{\rm ag(1)} = f_1$, for the second interface, $f_{\rm ag(2)} = f_1 + f_2$, and for the third interface, $f_{\rm ag(3)} = f_1 + f_2 + f_3$. With this onion skin [15] model, a multiple fluid, multiple interface configuration within a given cell is systematically "reconstructed" by transforming the configuration into a sequence of simplified and distinct two-fluid, one-interface representations. This simplification occurs via agglomeration of two of the fluids with one or more of the other fluids for each interface i present in the cell.

Besides the lack of an alternate homogeneous mix model, there are several acknowledged problems and weaknesses with the volume tracking onion skin model for multiple (> 2) fluids within a cell:

- the priorities p for each fluid are in general spatially and temporally dependent, but are instead arbitrarily specified as constants by the user;
- agglomerating volume fractions within a cell can degrade and misrepresent the actual interface configuration because of incorrect interface topology (normal) estimates;

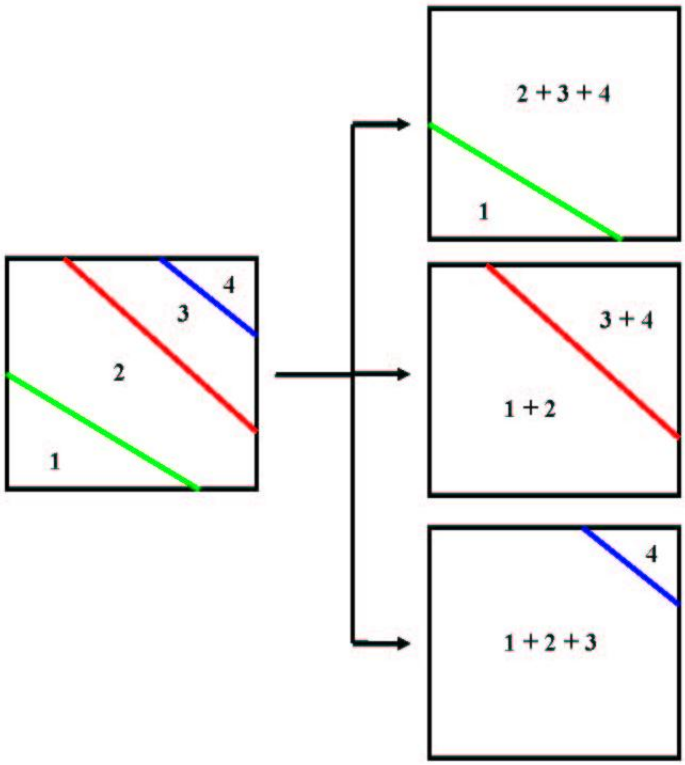


Figure 3.4: A cell occupied by four immiscible fluids separated from one another by three interfaces (left) is simplified into a sequence of three separate two-fluid, one-interface representations (right, top-to-bottom) by agglomerating two of the fluids with one or more of the other fluids for each step in the sequence. The agglomeration order depends upon specified fluid *priorities*, e.g., a *priority 1* fluid is treated first (right top), the *priority 2* fluid is agglomerated next (right middle), followed by the *priority 3* fluid. In this example, fluid one has priority one, two priority two, three priority three, and four priority four.

- there is nothing precluding the intersection of interfaces within a cell (other than brute force approaches used in [16]), which can lead to volume fluxes having the wrong sign; and

- extensions for the presence of "non-fluid" (rigid solid boundaries, solidified material, etc.) materials within a cell occupied by two or more fluids have not yet been formulated and tested.

As an example, for the second item above, if three fluids were to come in contact as a triple-point "T-junction" (each fluid interface orthogonal to one of the others), the second reconstructed interface will have a unit normal whose direction is as much as 50% in error. So in order to have a truly robust and accurate volume tracking representation for multiple flowing and rigid solid materials residing within computational cells, all of the above items can and must be addressed.

3.2.2 Interface Dynamics: Surface Tension

Surface tension at interfaces between fluids is represented by Equation 3.10 as a body force acting on nearby fluid. This approach, of modeling surface tension as a body force and so including it as a term in the Navier-Stokes Equation 3.2 was introduced by Brackbill et al. [3], and is referred to as the *Continuum Surface Force*, or CSF method. Since its introduction, many variations on the original method have been introduced. At the present time in TRUCHAS, the implementation of surface tension has not been generalized to the case of more than two fluids, the topic introduced in Section 3.2.1.5.

The surface tension force $\mathbf{f}_S$ of Equation 3.10 is decomposed into the sum of a normal $\mathbf{f}_S^N$ and a tangential $\mathbf{f}_S^T$ force component:

$$\mathbf{f}_S = \mathbf{f}_S^N + \mathbf{f}_S^T \tag{3.36}$$

with

$$\mathbf{f}_S^N = \sigma\kappa(\nabla f) \tag{3.37}$$

and

$$\mathbf{f}_S^T = (\nabla\sigma - \hat{\mathbf{n}}(\hat{\mathbf{n}}.\nabla)\sigma)\delta. \tag{3.38}$$

$\mathbf{f}_S^T$ is discretized at cell-centroids to achieve force balance between viscous stresses and capillary forces. $\mathbf{f}_S^N$ is discretized at faces and is treated together with the pressure gradient, in a similar way as the body force $\mathbf{f}_B$, to ensure proper balance of pressure gradient with surface tension force [17]. The curvature at face κ_f is obtained by simple interpolation of the cell-centered curvature κ. Next the techniques to compute the curvature κ are described.

The curvature κ of an interface is a function strictly of the unit normals $\hat{\mathbf{n}}$ to the interface:

$$\kappa = -\nabla \cdot \hat{\mathbf{n}}. \tag{3.39}$$

As a result, discretization of Equation 3.36 is reduced to an evaluation of normals. In the current implementation, we consider two convolution methods to compute smoothed normals. Note that the convolution methods are only used in the intermediate step of evaluating normals in the computation of the curvature. The computation of the normals used in the reconstruction of the interface planes is presented in Section 3.2.3.

1. The first approach follows the work of Rudman [18]. The normals are computed from smooth volume fractions. The smooth volume fractions $\tilde{f}$ are obtained by convolution of the discontinuous volume fractions with a Kernel K over a smoothing length d:

$$\tilde{f} = K * f, \tag{3.40}$$

where K is a cubic B-spline properly normalized and defined as

$$K(r,d) = \frac{1}{d^2} \begin{cases} \frac{40}{7\pi}\left(1 - 6\left(\frac{r}{d}\right)^2 + 6\left(\frac{r}{d}\right)^3\right), & \text{if } \frac{r}{d} \leq \frac{1}{2}; \\ \frac{80}{7\pi}\left(1 - \left(\frac{r}{d}\right)^3\right) & , \quad \text{if } \frac{1}{2} < \frac{r}{d} \leq 1; \\ 0 & , \quad \text{otherwise}, \end{cases} \tag{3.41}$$

with $r^2 = x^2 + y^2 + z^2$. The smoothed unit normals are then computed from $\tilde{f}$:

$$\hat{\tilde{\mathbf{n}}} = \frac{\nabla \tilde{f}}{|\nabla \tilde{f}|}. \tag{3.42}$$

2. The second approach follows the work of Williams [19]. The normals are obtained directly by convolving the volume fractions with kernel derivatives:

$$\hat{\tilde{\mathbf{n}}} = (K_x * f, K_y * f, K_z * f), \tag{3.43}$$

where

$$K_x = \frac{\partial K_6}{\partial x}, K_y = \frac{\partial K_6}{\partial y}, K_z = \frac{\partial K_6}{\partial z}, \tag{3.44}$$

and K_6 is the following kernel:

$$K_6(r,d) = \begin{cases} \frac{4}{\pi d^8}\left(d^2 - r^2\right)^3, & \text{if } \frac{r}{d} < 1; \\ 0 & , \quad \text{otherwise}. \end{cases} \tag{3.45}$$

Once the smoothed unit normals are evaluated, the curvature κ is calculated at cell-centers via a simple discretization of Equation 3.39:

$$\kappa \approx -\sum_{f=1}^{6} \frac{\hat{\tilde{\mathbf{n}}} \cdot \hat{\mathbf{n}}_f A_f}{V}. \tag{3.46}$$

$\hat{\tilde{\mathbf{n}}}$ is the unit normal to an interface, $\hat{\mathbf{n}}_f$ the unit normal to each cell face, A_f the area of each cell face, and V the volume of the cell.

3.2.3 Interface Topology

A normal vector to an interface between a material k and any number of other materials is strictly defined as the gradient of the volume fraction data

$$\mathbf{n} = \nabla f_k \text{ and } \hat{\mathbf{n}} = \frac{\nabla f_k}{|\nabla f_k|} \tag{3.47}$$

But the f_k are a discrete form of a discontinuous Heaviside function, and so estimates of the gradients tend to be very inaccurate. What is usually done, and what is implemented in TRUCHAS, is a spatial smoothing of the f_k field prior to the gradient evaluation, that leads to far better estimates of the normals.

The actual gradient calculation, at cell centers and at faces, is carried out via the least squares algorithms documented in Appendix A.

3.2.4 Property Evaluation

Knowing the new-time volume fractions for materials in every mesh cell permits the straightforward evaluation of the fluid density and viscosity throughout the mesh. The properties of each individual material may vary with the local temperature and species concentration; this dependence must be combined with the material volume fractions to arrive at fluid properties for each cell. The dependence of properties other than density is approximated by the general relation:

$$\Phi(\chi_i, T_i) = \sum_k c_{T,k} (T_i - T_{ref})^{e_{T,k}} + \sum_m c_{\chi,m} (\chi_i - \chi_{ref})^{e_{\chi,m}} \tag{3.48}$$

where $c_{T,k}$ and $c_{\chi,m}$ represent a coefficient of volume expansion associated with a temperature and solutal change respectively, T_{ref} is the reference temperature for the material, and χ_{ref} the reference solute concentration for the material.

The fluid density of each material can be evaluated in one of two ways. If the Boussinesq model is invoked, then individual material densities are independent of the local state, and the fluid density is simply a mixture value of the reference values. In this case, buoyant forces are treated in the body force evaluation. Otherwise, the full local state dependence is used throughout the flow solution.

A different formulation of the temperature and concentration dependence is used for the density in order to

correspond to standard terminology in the field. The density of each material is evaluated from:

$$\rho\left(\chi_i, T_i\right) = \rho_{ref} \cdot \left[1 + \sum_k c_{T,k}\left(T_i - T_{ref}\right)^{e_{T,k}} + \sum_m c_{\chi,m}\left(\chi_i - \chi_{ref}\right)^{e_{\chi,m}}\right] \tag{3.49}$$

3.2.5 Momentum Equation

Our implementation is on an unstructured hexahedral mesh, with the primary variables $\mathbf{u}$ and p located at cell centers. To assess solenoidality, we also calculate a velocity field $\mathbf{u}_f$ at cell face centroids.

As described above, we discretize Equation 3.2 to first order in time, and by introducing an interim "predicted" velocity $\mathbf{u}^\star$, divide the resulting equation in two:

$$\frac{\rho^{n+1}\mathbf{u}^\star - \rho^n\mathbf{u}^n}{\Delta t} = -\nabla\cdot(\rho\mathbf{u}\mathbf{u})^n + \nabla\cdot(\mu^{n+1}(\nabla\mathbf{u} + \nabla^T\mathbf{u})) + \mathbf{f}_S^{n+1} + \mathbf{f}_D^{n+1} - \nabla P^n + \mathbf{f}_B^n \tag{3.50}$$

$$\frac{\rho^{n+1}\mathbf{u}^{n+1} - \rho^{n+1}\mathbf{u}^\star}{\Delta t} = -\nabla\delta P^{n+1} + \mathbf{f}_B^{n+1} - \mathbf{f}_B^n \tag{3.51}$$

Equation 3.50 is a relation for $\mathbf{u}^\star$, referred to as the predictor step. Equation 3.51 is termed the projection step. Combining Equation 3.50 and Equation 3.51 exactly reproduces the time discretization of Equation 3.2; no additional approximation results from this decomposition, which is made simply for computational convenience.

Equation 3.50 and Equation 3.51 are evaluated at cell centers. $\mathbf{f}_S$, $\mathbf{f}_D$ and $\mathbf{f}_B$ represent the surface tension, drag and body forces respectively. The unspecified time level of the viscous stress will be clarified below (Section 3.2.8).

The full solution of the momentum equation is more complex than implied by these equations because we employ face velocities to impose the solenoidal condition on the velocity field, as will become clear in the sequel.

3.2.6 Predictor Step

Solution of Equation 3.50 may proceed by either of two paths, depending upon the time level associated with the viscous stress. If a fully explicit approximation is used, evaluation of $\mathbf{u}^\star$ can proceed pointwise or in parallel. If not, solution of a coupled set of linear equations is necessary. This is accomplished by the

methods described in Appendix C. The sections below describe each of the terms on the right hand side in turn.

3.2.7 Momentum Advection

Momentum advection is evaluated by using the volume tracking results to calculate $\nabla \cdot (\rho \mathbf{u}\mathbf{u})^n$. Consider Figure 3.5, that illustrates the advection of material across the right face of a cell containing an interface between two fluids. The advected volume is:

$$\delta V_f = \Delta t A_f \mathbf{u}_f \cdot \hat{n}_f \tag{3.52}$$

where $\mathbf{u}_f \cdot \hat{n}_f$ is the component of the solenoidal face velocity $\mathbf{u}_f$ normal to the face, and A_f is the face area. Discretizing the advection term of Equation 3.2, we obtain:

$$\Delta t \int \nabla \cdot (\rho \mathbf{u}\mathbf{u})^n dV \approx \sum_f \delta V_f \langle \rho \mathbf{u} \rangle_f^n \tag{3.53}$$

The volume tracking algorithm calculates the subvolumes $\delta V_{k,f}$ within the flux volume δV_f (see Figure 3.5), and we introduce $f_{k,f}$ to represent the volume fraction of δV_f associated with a particular material k:

$$f_{k,f} = \frac{\delta V_{k,f}}{\delta V_f} \tag{3.54}$$

Multiplying the $\delta V_{k,f}$ by corresponding densities yields the mass of each material k leaving a cell across face f:

$$M_{k,f} = \rho_k \delta V_{k,f} \tag{3.55}$$

and we designate the total mass leaving a cell across face f as $M_f \equiv \sum_k M_{k,f}$. Equation 3.53 may then be written in a way that is wholly consistent with mass advection:

$$\Delta t \int \nabla \cdot (\rho \mathbf{u}\mathbf{u})^n dV \approx \sum_f \sum_k M_{k,f} \langle \mathbf{u} \rangle_f^n = \sum_f M_f \langle \mathbf{u} \rangle_f^n \tag{3.56}$$

Note that $\langle \mathbf{u} \rangle_f^n$ is the estimate of the advected momentum per unit mass. For the purposes of this report we have chosen this to be a simple upwind value, producing a first order accurate approximation.

3.2.8 Momentum Diffusion

The current version of TRUCHAS offers a model for laminar Newtonian viscous stress and a simple algebraic turbulence model. If turbulence is modeled, the total stress tensor is determined by the velocity gradients and the effective viscosity as given in Equation 3.5.

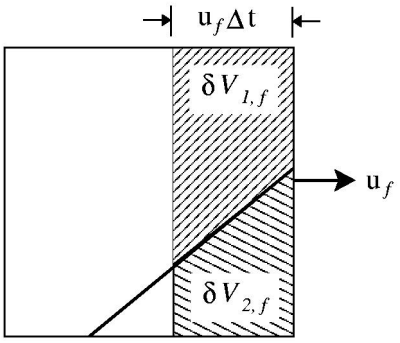


Figure 3.5: The advection of material across the face of a cell containing an interface between two fluids. The volume tracking algorithm calculates the individual flux volumes $V_{1,f}$ and $V_{2,f}$.

Viscous forces are incorporated into the predictor estimate of the cell-centered velocity field by averaging time n and time $*$ values of the velocity field. The selection of the averaging is controlled by the user through an input parameter `VISCOUS_IMPLICITNESS`. The explicit approximation eliminates the need for solving a system of equations, while more implicit approximations are stable for larger time steps.

The net viscous stress on the control volume is calculated by applying the divergence theorem to the volume integral of the local stress. This reduces to a sum of the dot product of the face normal vector with the local velocity gradient multiplied by the face area:

$$\vec{F}_v = \sum_f \mu_f A_f [\hat{\mathbf{n}}_f \cdot \left(\nabla \mathbf{u} + \nabla^{\mathbf{T}} \mathbf{u}\right)] \tag{3.57}$$

The velocity gradient can be calculated by either the weighted least squares method, or by an orthogonal approximation, as selected by the user. (See the description of `USE_ORTHO_FACE_GRADIENT` in the `Reference Manual`.) Appendix A (Discrete Operators) describes the least squares method in a general context. The tensor nature of the velocity gradient is simply handled component by component. The velocity gradient is first order accurate, resulting in second order errors in the viscous stress.

The fluid viscosity is harmonically averaged to face centroids from cell centered values using an orthogonal approximation. The cell centered values of viscosity are the sum of the fluid-weighted, temperature and solute concentration dependent laminar values and the product of the eddy viscosity and the fluid density.

Boundary conditions are enforced on all faces where they are specified. This is performed by modifying the velocity gradient after its initial evaluation by least squares at those locations. The boundary conditions are not directly incorporated into the weighted least squares evaluation, and therefore only affect the velocity

gradient at the faces at which they are applied.

3.2.9 Transfer the cell-centered Velocity to Faces

To facilitate the volume tracking algorithm (Section 3.2.1) we require a set of face velocities that are solenoidal with respect to the basic conservation cells. To ensure this, we compute the face velocities from the cell centered velocities, then impose the solenoidality condition (Equation 3.12).

To suppress spatial oscillations in the pressure and velocity, we apply a modification of the Rhie-Chow [6] procedure. That is, we interpolate a modified version of $\mathbf{u}^\star$ that is augmented by the cell-centered approximation to the difference between the body force and the pressure gradient (multiplied by $\Delta t/\rho$), evaluated at time n. See the first term on the right hand side of Equation 3.58.

The interpolation is done using a least squares linear reconstruction technique similar to that of Barth [20], as we assume that the normal component of velocity does not vary discontinuously near an interface. This is the $c \rightarrow f$ step in Equation 3.58. The resulting stencil typically includes several cells in the region of the face under consideration. Density, on the other hand, varies discontinuously across an interface so we limit the size of this stencil by calculating a face density using only the two cell densities adjacent to the face.

After interpolation, a face-centered approximation to the difference between the body force and the pressure gradient (multiplied by $\Delta t/\rho$) is added to the face velocity, to obtain the time level $*$ face velocity. That is:

$$u_f^* = \left\langle u_c^* + \delta t \left\langle \frac{1}{\rho_f}(\nabla_f P - \mathbf{f}_B) \right\rangle_{f\rightarrow c}^{n} \right\rangle_{c\rightarrow f} - \frac{\delta t}{\rho_f^{n+1}} \left(\nabla_f P^n - \mathbf{f}_B^{n+1}\right) \tag{3.58}$$

Here, the operator $f \rightarrow c$ means averaging the face values of this quantity, in the same fashion as used in Section 3.2.11.)

This combination of the body force and pressure gradient permits numerically balancing the body force with a pressure gradient, if the curl of the body force is zero. (Gravitational forces are a trivial example of a zero curl body force.) Cancellation is critical to the correct computation of hydrostatic (or nearly hydrostatic) situations, to avoid numerical oscillations which prevent the simulation from reaching steady state.

3.2.9.1 Transfer of cell-centered velocities to faces when using orthogonal operators

In the event that ortho-approximations are used for the discrete operators (either because the user has explicitly requested it, or because TRUCHAS has determined that the mesh is orthogonal) a different approach

is taken in balancing the pressure and gravitational terms. In this case the pressure gradient and the gravity force are combined (in Equation 3.50) by including the body force due to gravity, $\mathbf{f}_B^n$, in the pressure gradient term. The continuous form of the pressure term, which is now the gradient of the dynamic pressure, is then:

$$\nabla(P - (\rho + \Delta\rho)\vec{g} \cdot \vec{x}) \tag{3.59}$$

The time n value of Equation 3.59 is determined from an average of the previous time's face values. As described in the previous section the predicted face velocity is calculated using the most recent face centered value of the pressure gradient. In this case Equation 3.59 is evaluated on a face.

In general we need to consider that two neighboring cells have different densities so that the discrete (normal) gradient of the dynamic pressure across cell face f for cell i is:

$$\left(\frac{\partial P}{\partial n}\right)_i^f = \frac{(P_i - (\rho_i + \Delta\rho_i)\vec{g} \cdot \vec{\Delta x}_i - (P_n - (\rho_n + \Delta\rho_n)\vec{g} \cdot \vec{\Delta x}_n))}{\Delta l} \frac{\vec{\Delta l}}{\Delta l} \cdot \vec{n} \tag{3.60}$$

where P_i/P_n, ρ_i/ρ_n, and $\Delta\rho_i/\Delta\rho_n$ are the cell centered pressure, density, and buossinesq density variation for cell i and its neighbor n. $\vec{\Delta x}_i/\ \vec{\Delta x}_n$ is the distance from the cell centroids of cell i and its neighbor n. $\vec{g}$ is the gravitational acceleration, $\vec{\Delta l}$ is the vector distance between the cell centroids, Δl is the magnitude of the distance between the cell centroids and $\vec{n}$ is the cell face normal. The densities and their deviates are evaluated at the advanced time in this step.

The equation for interpolating the cell centered velocities to the faces then becomes

$$u_f^* = \left\langle u_c^* + \delta t \left\langle \frac{1}{\rho_f} \nabla_f P^d \right\rangle_{f\to c}^n \right\rangle_{c\to f} - \frac{\delta t}{\rho_f^{n+1}} \left(\nabla_f P^d\right)^n \tag{3.61}$$

dotted with the cell face normal since we are only concerned with the normal component of the velocity. The pressure gradient $\nabla_f P^d \cdot \vec{n}$, where P^d is the dynamic pressure, is calculated from Equation 3.60.

By following this process the static pressure is accounted for exactly, thus removing errors in the contribution of the static component to the total pressure. This is particularly important when the orthogonal operators are used on non-orthogonal meshes. Without this procedure one would see fluid erroneously moving about in a static pressure head calculation.

3.2.10 Projection

Equation 3.51 relates $\mathbf{u}^{n+1}$ to $\mathbf{u}^{\star}$; combining Equation 3.51 with Equation 3.12 yields:

$$\nabla \cdot \frac{\nabla \delta P^{n+1}}{\rho^{n+1}} = \nabla \cdot (\frac{\mathbf{u}^{\star}}{\Delta t}) \tag{3.62}$$

We solve Equation 3.62 for δP^{n+1} (using the techniques of Appendix C), and complete the timestep by evaluating $\mathbf{u}^{n+1}$ via Equation 3.51.

Divergences are calculated by summing over cell faces, and $\nabla \delta P_f$ is calculated from a stencil corresponding to that of the density interpolation to faces. Equation 3.51 is then used to calculate a solenoidal (to the precision required by the projection solution) face velocity field:

$$\mathbf{u}_f^{n+1} = \mathbf{u}_f^{\star} - \Delta t(\frac{\nabla \delta P_f^{n+1}}{\rho_f^{n+1}}) \tag{3.63}$$

3.2.11 Adjust Cell Centered Velocity for Pressure Gradient

The final step is to interpolate $\nabla \delta P_f^{n+1}/\rho_f^{n+1}$ to cell centers in order to obtain $\mathbf{u}^{n+1}$ from $\mathbf{u}^{\star}$ via Equation 3.51.

Note that only the dynamic component of the pressure gradient is used in this step. Subtracting the hydrostatic component from pressure gradient before averaging permits the numerical cancellation of body force and pressure gradient in both the face centered and cell centered velocity results.

The average is calculated by weighting the value at each face by the face area normal to each coordinate direction.

When orthogonal discrete operators are being used we interpolate $\nabla \delta (P_f^d)^{n+1}/\rho_f^{n+1}$.

3.2.12 Flow Past Solid Material

Some mesh cells may occasionally contain a sharply defined partial volume of solid material, either when a solid mold material is initially defined to occupy a portion of the computational domain not exactly aligned with mesh cells, or when a pure material undergoes solidification, such that the interface between liquid and solid is a well defined moving front. In either case, the flow solver must then deal with cells that contain less than an entire cell volume of fluid.

The TRUCHAS approach is as follows. Cell faces are defined in a binary way to be entirely closed to flow, or not. Cell faces are deemed to be "closed" only if at least one of the two immediately neighboring cells is entirely composed of solid material. In that case, the face velocity of the cell is set to zero, and the face pressure gradient is no longer included in the pressure solution..

Any face between two cells that both contain at least a partial cell volume of fluid is deemed to be "open", and the code solves for a velocity and a pressure gradient. No attempt is made to account for partial face areas open to flow, because of the complex geometry associated with an unstructured hexahedral mesh and also because cell faces may not be planar.

3.2.13 Flow Through Porous Media

Solidification of many materials, including most metals, proceeds by the formation of solid dendrites embedded within a liquid matrix. This process generates a solid morphology that provides only tortuous paths for fluid to traverse, leading to significant drag forces on the liquid component. These drag forces are often non-isotropic which can lead to interesting macroscopic features (such as channels) in the solidifying flow.

TRUCHAS uses a porous medium model to represent such drag forces. The model assumes that the drag force is linearly dependent on the fluid superficial velocity (Darcy law). Each velocity component is treated independently, and the coefficients (permeability) in each direction are independent. We have not implemented a tensor permeability because of the complexity of such a model and the lack of information upon which to base it.

The specific form chosen for the drag is the Carman-Koseny relation [21]:

$$\vec{F}_{drag} = -\overleftrightarrow{C} \cdot \frac{(1-f)^2}{f^3} \vec{u}^{n+1} \tag{3.64}$$

where $\overleftrightarrow{C}$ is the directional permeability and f the fraction of the cell volume occupied by fluid. The directional permeability is a diagonal tensor whose elements are evaluated as volume weighted averages of the solid material permeabilities within each cell.

3.2.14 Treating Some Fluids as Void

Because both pressure gradients and momentum fluxes are often much smaller in gas regions of the fluid domain than the liquid regions, a void model can be used to simplify the equations to be solved. We permit

a special type of fluid in TRUCHAS to identify regions we wish to model as void by prescribing that the fluid density is zero for the void material. This has several effects on the solution:

Void Cells: Cells that contain only materials of zero density have zero momentum and a fixed pressure. Therefore, it is unnecessary to solve any of the flow equations in such cells. All the components of the fluid velocity are set to zero in void cells. The temperature in void cells is set to an input value, primarily for visualization purposes, but does not affect the thermal solution elsewhere in the domain.

Mixed Cells: Cells that contain a mixture of void material and one or more other fluid materials are treated differently than cells that contain no void. The difference derives from the need to model the "compressibility" of void materials. That is, we may not wish to treat such materials as having a constant density. The compressibility of void material is expressed as a modification of the continuity equation (Equation 3.12), which becomes:

$$\nabla \cdot \mathbf{u} = \xi \frac{\partial P}{\partial t} \tag{3.65}$$

where ξ is the compressibility of the local fluid. Although vacuum is infinitely compressible, it has been found that using a finite value for the compressibility leads to a more robust numerical procedure; therefore TRUCHAS permits the specification of this through an input parameter.

We wish to transition smoothly as the void fraction of a cell diminishes toward zero. Therefore, we choose to determine the compressibility of a cell as a weighted average of the materials in the cell. Since all materials other than void have no compressibility, this leads to simply multiplying the compressibility of the void material by the fraction of the fluid in the cell that is void.

We also make use of the thermodynamics of isentropic compression of a perfect gas to relate the compressibility to an effective "sound speed" of the void through the relation:

$$\xi \equiv -\frac{1}{\rho c^2} \tag{3.66}$$

where c represents the sound speed, which is the parameter chosen as input for TRUCHAS. The density chosen for this relationship is the average fluid density in the cell.

This model is sometimes referred to as the "void collapse" model in TRUCHAS. This term expresses the idea that void compressibility eliminates a problem that existed in previous versions of the program. Since those versions treated void materials as incompressible in any cell that contained other fluid, it was not possible for internal void regions (bubbles) to collapse completely. Filling simulations frequently showed residual voids, which led to considerable additional problems with the thermal solution.

The present model relies on the specification of a “reasonable” sound speed, that must currently be determined by computational experience. Research is continuing on ways to reduce the arbitrary nature of this model.

Chapter 4

Heat Transfer and Phase Changes

The following chapter presents the heat transfer and phase change algorithms incorporated into TRUCHAS. The heat transfer algorithm solves the enthalpy advection equation with sources implicitly. The phase change algorithm assumes binary temperature-concentration phase diagrams but can employ multiple phase transformation types. The heat transfer and phase change section is responsible for calculating the enthalpy, solutal concentration and solid volume fractions given fluid velocities and chemical, electromagnetic and user specified heat sources.

4.1 Physics

4.1.1 Assumptions

We list the primary assumptions of the heat transfer and phase change modeling below.

- Local thermodynamic equilibrium, e.g., $T_1 = T_2 = T_m = T$ for all materials m in a given cell.
- No solid movement
- Scalar, isotropic heat diffusion
- Use of binary temperature-concentration phase diagrams whose equilibrium lines are approximated by straight lines.
- The specific enthalpy (per volume, or per mass) does not depend on pressure and only depends on temperature and concentration

4.1.2 Material Properties

The concept of a "material" is extended to include "phases." Solid Cu and liquid Cu are therefore different "materials." Each cell may contain $nmat$ materials at thermodynamic equilibrium.

The density of each material is assumed constant and the average density of the cell is given by:

$$\rho_{cell} = \sum_{m}^{nmat} \rho_m f_m \tag{4.1}$$

where f are the volume fractions of all materials in the cell and:

$$\sum_{m=1}^{nmat} f_m = 1 \tag{4.2}$$

In the current implementation, materials that are part of a series of phase transformations must all have the same density. The specific heat capacity at constant pressure (per unit mass) of each material is represented as a polynomial in temperature T and composition c:

$$C_P(T) = \sum_{i=0}^{imax} C_i T^{e_i} + \sum_{j=0}^{jmax} D_j c^{e_j} \tag{4.3}$$

where the indexes i and j, C and D are real coefficients and $e_i, e_j \neq -1$ are real exponents.

A typical form of the temperature depended heat capacity, consistent with the commercial databases, is:

$$C_P(T) = A + BT + CT^2 + DT^{-2} \tag{4.4}$$

The heat capacity is related to the specific enthalpy (per unit mass) by:

$$C_P(T) = \left(\frac{\partial h}{\partial T}\right)_P \tag{4.5}$$

That allows for the integration of the heat capacity to obtain the specific enthalpy:

$$h(T) = h_{\text{ref}} + \int_{T_{\text{ref}}}^{T} C_P(\tau) d\tau \tag{4.6}$$

where the superscript 'ref' is used to describe the reference temperature and enthalpy. The reference states are internally chosen as the lowest temperature of stability for a certain material and the corresponding enthalpy. As an example, for a liquid, the melting temperature is used as reference (see Figure 4.1). In

the case of binary alloys, the reference is the temperature corresponding to the liquidus line, at the specified concentration. The most stable solid phase is referenced to zero, both in temperature and enthalpy. Although the latent heat is temperature dependent, the parameter used as input is the latent heat L^L at the liquidus temperature. For an isothermal transformation, $T^S = T^L$ and $L^S = L^L$. In the mushy zone Equation 4.5 defines the "effective" heat capacity.

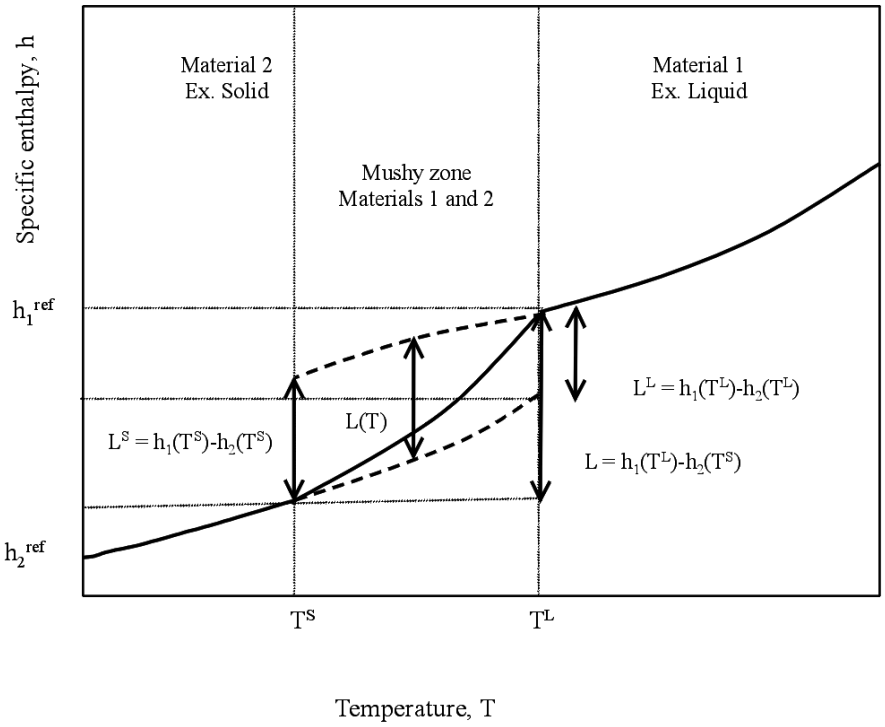


Figure 4.1: Typical specific enthalpy of an alloy as function of temperature. The enthalpy in the mushy zone depends on the phase transformation model.

The integration of the typical form of the heat capacity Equation 4.4 leads to:

$$h(T) = h_{\text{ref}} + A(T - T_{\text{ref}}) + \frac{B}{2}(T^2 - T_{\text{ref}}^2) + \frac{C}{3}(T^3 - T_{\text{ref}}^3) - D\left(\frac{1}{T} - \frac{1}{T_{\text{ref}}}\right) \tag{4.7}$$

The total enthalpy H of a cell of volume V_{cell} is given by:

$$H_{cell}(T) = V_{cell} \sum_{m=1}^{nmat} \rho_m f_m(T) h_m(T) \tag{4.8}$$

The cell properties are referenced to zero, for both temperature and enthalpy, to avoid the ambiguity created by various references in multi-material cells. For constant heat capacity, various specific (per unit mass)

properties of the "fully homogenized" material in a cell are internally calculated and used for checking thermodynamic consistency:

$$h_{cell}(T) = \frac{\sum_m h_m(T)\rho_m f_m}{\rho_{cell}} \tag{4.9}$$

$$C_{Pcell} = \frac{\sum_m C_{Pm}\rho_m f_m}{\rho_{cell}} \tag{4.10}$$

$$h^{\text{ref}}_{cell} = \frac{\sum_m \left(h^{\text{ref}}_m - C_{Pm}T^{\text{ref}}_m\right)\rho_m f_m}{\rho_{cell}} \tag{4.11}$$

4.1.3 Phase Diagrams

Only binary temperature-concentration phase diagrams are allowed for now. Multi-component systems will be included in future versions of the code. The binary diagrams can feature complete solubility or eutectic points. Figure 4.2 shows a typical diagram of a generic system with complete solubility both in the liquid (L) and solid solution (SS). The melting temperatures of the two components are T^M_A and T^M_B. The equilibrium lines are approximated by straight lines. At a given temperature T_1, after thermodynamic equilibrium was achieved, the concentrations in the two phase are given by x^L and x^S. For a given concentration x_0, the corresponding temperatures are T^L and T^S.

Given the straight line approximation, the slopes of the liquidus and solidus lines are:

$$S^L = \frac{T^L - T^M_A}{x_0} \tag{4.12}$$

$$S^S = \frac{T^S - T^M_A}{x_0} \tag{4.13}$$

Both positive and negative slopes are allowed (see Figure 4.3). The partition coefficient is defined as:

$$K = \frac{x^S}{x^L} = \frac{S^L}{S^S} \tag{4.14}$$

The eutectic points (Figure 4.4) are define by composition x^E and temperature T^E. At the eutectic temperature the transformation is isothermal. A limitation of the current implementation is that for x_0 between the solubility limits, the concentration is fixed below the temperature T^E, and the decomposition $SS_1 + SS_2$ is not captured.

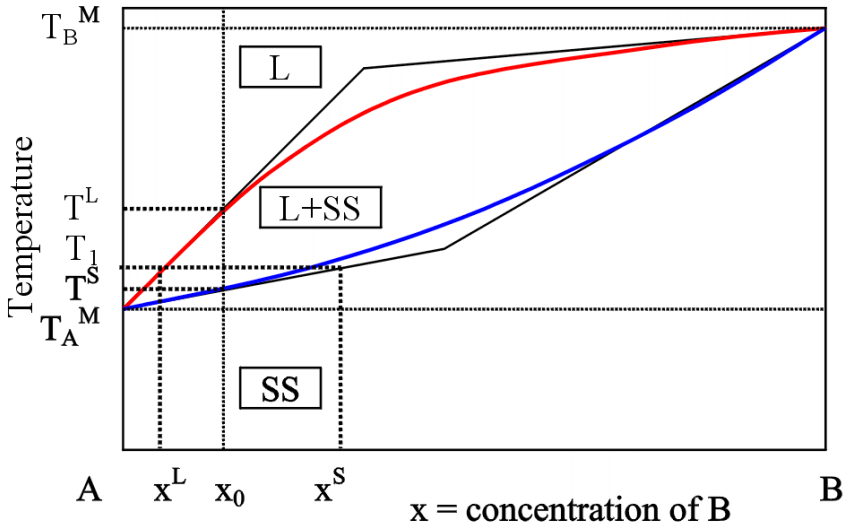


Figure 4.2: Binary system with complete solubility and positive liquidus slope.

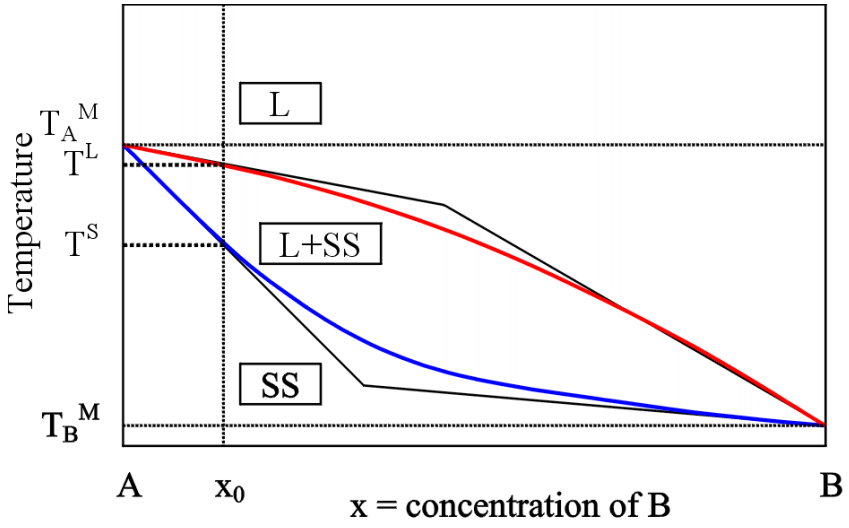


Figure 4.3: Binary system with complete solubility and negative liquidus slope

4.1.4 Phase Changes

Multiple, consecutive phase transformations are allowed in Truchas. Besides the isothermal transformation, the “non-isothermal” type is defined by the specific enthalpy of an alloy as a function of temperature (see Figure 4.1). This is useful to describe multi-component alloys for which the phase diagram is unknown.

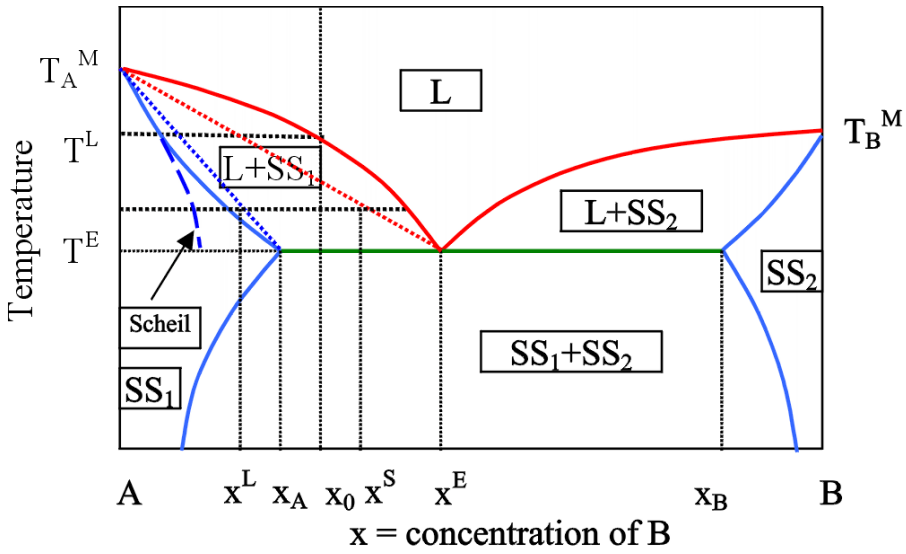


Figure 4.4: Binary system with eutectic

Several phase transformation models are implemented to describe binary alloys. The models provide the volume fractions at a given temperature and are described in more detail below.

4.1.4.1 Lever rule

In this model the diffusivity of the solute is assumed to be infinite in both the liquid and the solid. The system equilibrates instantaneously and the volume fraction of the solid at temperature T is:

$$f^S = \left(1 - \frac{x_0}{x^L}\right) \frac{1}{1-K} \tag{4.15}$$

In the present straight-line approximation, we have

$$\frac{x_0}{x^L} = \frac{T^L - T_A^M}{T - T_A^M} \,. \tag{4.16}$$

4.1.4.2 Scheil

The Scheil model preserves the infinite diffusivity in the liquid but assumes that the solute does not diffuse at all in the solid. The volume fraction of the solid is given by:

$$f^S = 1 - \left(\frac{x^L}{x_0}\right)^{\frac{1}{K-1}} \tag{4.17}$$

For the eutectic diagram shown in Figure 4.4, the dashed line represents the average concentration in the solid during a solidification process. However, at any fixed temperature, the concentration of the new formed solid is given by the equilibrium solidus line (blue).

4.1.4.3 Clyne and Kurz

This model allows for a finite value of the diffusivity of the solute in the solid phase [22]. The diffusivity in the liquid is still infinite. The model features a control parameter, relating the diffusivity of the solute in the solid phase D_B^S, the solidification time Δt, and the dendrite arm spacing λ:

$$\alpha = \frac{D_B^S \, \Delta t}{(\lambda/2)^2} \tag{4.18}$$

The parameter α can take any positive value and is mapped onto the [0, 0.5] interval by the function:

$$F(\alpha) = \alpha \left(1 - e^{-\frac{1}{\alpha}}\right) - 0.5 \, e^{-\frac{1}{2\alpha}} \, . \tag{4.19}$$

Finally, the volume fraction of the solid is calculated as:

$$f^S = \frac{1}{u} \left[1 - \left(\frac{x^L}{x_0} \right)^{\frac{u}{K-1}} \right] \, , \tag{4.20}$$

with

$$u = 1 - 2KF(\alpha) \, . \tag{4.21}$$

Here, K is the partition coefficient (Equation 4.14). We note that the lever rule corresponds to $F(\alpha) = 0.5$, whereas the Scheil equation corresponds to $F(\alpha) = 0$.

4.1.4.4 Volume Change During Phase Change

In case of phase changes that involve a volume change we compute the volume change due to phase change by conserving the mass of the cell. Let us say that there is a cell which has a mass M_1 defined by materials of volumes ${V_1}^i$ and densities ρ_1^i. Let us say that the enthalpy of this cell changes resulting in a phase change. This changes the volumes of the materials to ${V_2}^i$ and the densities to ρ_2^i resulting in a mass change to M_2. If we constrain the mass of the cell after phase change to be equal to the mass of the cell before phase change, this sets up the following equations:

$$\begin{aligned} & M_1 = M_2 = M \\ \Rightarrow \quad & \left(\rho_1^H f_1^H + \rho_1^L f_1^L\right) \left({V_1}^H + {V_1}^L\right) = \left(\rho_2^H f_2^H + \rho_2^L f_2^L\right) \left({V_2}^H + {V_2}^L\right) \\ \Rightarrow \quad & \left({V_2}^H + {V_2}^L\right) = \left({V_1}^H + {V_1}^L\right) \left[\frac{\rho_1^H f_1^H + \rho_1^L f_1^L}{\rho_2^H f_2^H + \rho_2^L f_2^L}\right] \end{aligned} \tag{4.22}$$

where the superscript H refers to the high temperature phase and the superscript L refers to the low temperature phase involved in the phase change, the subscript 1 refers to values before the phase change took place and the subscript 2 refers to the final values after the phase change, and $f^H + f^L = 1$ for both states 1, and 2.

Using Equation 4.22 we can now calculate the change in total volume of the cell as:

$$
\begin{aligned}
V_{cell}^1 &= \sum_i {V_1}^i = \sum_{i \neq \{H,L\}} {V_1}^i + \left({V_1}^H + {V_1}^L\right) \\
V_{cell}^2 &= \sum_i {V_2}^i = \sum_{i \neq \{H,L\}} {V_2}^i + \left({V_2}^H + {V_2}^L\right) \\
\Rightarrow \quad V_{cell}^2 &= V_{cell}^1 + \left[\frac{\rho_1^H f_1^H + \rho_1^L f_1^L}{\rho_2^H f_2^H + \rho_2^L f_2^L} - 1\right]\left({V_1}^H + {V_1}^L\right)
\end{aligned}
$$

This gives us a change in volume of the cell as:

$$\Delta V_{cell} = \left[\frac{\rho_1^H f_1^H + \rho_1^L f_1^L}{\rho_2^H f_2^H + \rho_2^L f_2^L} - 1\right]\left({V_1}^H + {V_1}^L\right) \tag{4.23}$$

and a change in density of the cell as:

$$\Delta \rho_{cell} = M\left(\frac{1}{V_{cell}^2} - \frac{1}{V_{cell}^1}\right) \tag{4.24}$$

Note that if the densities of other materials in the cell were temperature dependent, then that would be reflected in this equation as well.

4.1.5 Boundary and Initial Conditions

4.1.5.1 Radiative Boundary Conditions

Two types of radiative boundary conditions are implemented in TRUCHAS:

- Radiation to an ambient temperature
- Viewfactor based enclosure radiation model

Radiation to an ambient Temperature
This boundary condition replaces the flux on boundary surfaces using the Stefan-Boltzmann law and a time varying ambient temperature. The flux leaving any surface calculated thus can be written as:

$$q = -\epsilon\sigma(T_f^4 - T_o^4) \tag{4.25}$$

where T_f is the (absolute) temperature at the boundary face, ϵ is the emissivity, T_o is the (absolute) ambient temperature, and σ is the Stefan_Boltzmann constant. Since TRUCHAS uses cell based temperatures, the temperature at the face is calculated by equating the flux leaving the surface due to radiation to the conductive flux from the center of the cell. This is achieved by solving the following equation:

$$k(T_f - T_c) - \epsilon\sigma(T_f^4 - T_o^4) = 0 \tag{4.26}$$

where k is the conductivity of the cell, T_c is the temperature at the cell temperature, and all other symbols are as defined before. The above equation is solved for every face that participates in the boundary condition at every iteration of the solver.

Viewfactor based enclosure radiation model
TRUCHAS also supports a more sophisticated radiation model for enclosures within the simulation domain where radiation leaving one face might arrive at another face within the system. In this case we use a viewfactor based enclosure radiation model. This model works by first calculating the fraction of energy leaving any given surface within the enclosure and arriving at any other surface therein, also called the viewfactors. We use the Chaparral package from Sandia National Laboratory to calculate the viewfactors, and additional details on the calculation of viewfactors can be found therein.

Once we have the viewfactors, we calculate the flux q_i at an external boundary face i using the temperatures on all other faces j and the ambient temperature as**:**

$$q_i = -\epsilon_i\sigma T_i^4 - (1 - \rho_i)\sum_j F_{ij}q_j \tag{4.27}$$

where ϵ_i is the emissivity of face i, σ is the Stefan-Boltzmann constant, T_i is the absolute temperature at the boundary face, F_{ij} is the view factor of face j as seen from face i, and the sum on j is over all faces in the system and includes a special term for the contribution from the ambient environmant.

This sets up a linear system of equations that is solved for the fluxes, q_i. As with the simple radiation model, the face temperatures can be found by nesting this solve within a second loop that adjusts T_f based on the flux from each face i to the center of the corresponding cell.

Since this calculation is done on every iteration of the solver for every timestep, it has the capability to slow down the heat transfersolution. To speed the calculation, we allow the user the option of choosing from the following two approximations:

- Use cell temperatures for T_f

- Use q based on previous timestep cell temperatures

The first of these approximations, using cell temperatures instead of face temperatures, eliminates the need for the nested solve for face temparatures, while the second approximation allows us to calculate the radiative fluxes once every time step and impose a flux boundary condition on the corresponding faces. Both these approximations are independent of each other and can be used based on the timestep, size of cells in the system, and the overall accuracy of the solution desired.

4.1.6 Conservation Law

The heat transfer algorithm solves for the mixture specific volume enthalpy h and mixture density ρ via the following conservation law,

$$\frac{\partial(\rho h)}{\partial t} + \nabla \cdot \left(\rho^L h^L f^L \mathbf{u}\right) - \nabla \cdot (\kappa \nabla T) = p \tag{4.28}$$

where f^L and h^L are the liquid volume fractions and the liquid specific volume enthalpy respectively (note that as solid is not mobile, only the liquid component of the enthalpy is advected). Body sources are represented by p. In the current version of the code these body sources are due to eddy current induced Joule heating, chemical exothermal reactions and to a user specified Gaussian source/sink.

4.1.7 Boundary Conditions

The boundary conditions implemented into the code are imposed on the temperature field and include Dirichlet and Neumann. It is also possible to specify a heat transfer coefficient, htc which will set the magnitude of thermal flux between two materials. The flux is set to be proportional to htc and to the temperature difference on opposite sides of the material interface: $flux = htc(T_+ - T_-)$. Convective cooling at a surface is implemented as a special case of the heat transfer coefficient boundary condition with T_- set to the ambient temperature outside the cooled part.

4.1.8 Interaction With Other Physics

The interactions of the heat transfer and phase change solution are primarily with the flow solution and to a lesser extent, the chemistry and electromagnetic solutions. Given the fluid velocity, this section calculates the mixture specific volume enthalpy due to flow advection, chemistry and electromagnetic heat sources.

It also calculates new temperatures, solutal concentrations and solid fractions which feedback into the flow solution as changes in material properties.

4.2 Heat Transfer Algorithm

The heat transfer algorithm in Truchas is based on a finite volume discretization of the enthalpy formulation of the energy conservation. The finite volume approach ensures that the numerical scheme is locally and globally conservative, while the enthalpy formulation allows treatment in a straightforward and unified manner many possible phase change mechanisms. The phase boundary in the enthalpy method is not tracked, but is captured via the continous solid/liquid fraction fields. A finite difference variant of the algorithm is described in a paper by Knoll, Kothe and Lally [23].

To solve Equation 4.28 an implicit treatment is employed for the diffusion and temporal terms, a semi-implicit treatment for the advection term and an explicit treatment for the source terms. This treatment is employed because the stability restrictions on the diffusion term are more severe than those on the advection or source terms. The sources currently allowed in the code are Joule heating, due to eddy currents flowing through the melt induced by an exterior induction coil (see Chapter 7), the power source of chemical nature (see Chapter 5), and a Gaussian source/sink as specified in the `HEAT SOURCES NAMELIST` in the `Reference Manual`.

4.2.1 The Discrete Equations and the Non-linear Residual

The non-linear residual given below in Equation 4.29 is used in this algorithm to both evaluate the next iterate of the Newton iteration and to evaluate the Jacobian · vector products during the linear solve with conjugate gradient-like Krylov iteration (see the chapter on Non-linear solution methods). The non-linear residual vector is *ncells* long, where *ncells* is the number of cells in the simulation. In every cell, upon convergence of the current non-linear iteration, the residual $F({}^kH_{n+1})$ should fall below a specified threshold. Here, k is the non-linear iteration index, $n+1$ is the index of the time level at which the solution is sought, and H is the total enthalpy in the cell of interest. During the iteration, the residual is non-zero and is written down explicitly with the use of local conservation of energy (Equation 4.28). The terms in Equation 4.29 are all in finite-volume form just as they appear in the code. The terms in curly brackets indicate a discrete approximation of the enclosed operator.

$$\left\{\frac{\partial H}{\partial t}\right\}_{n+1} + \left\{\sum_{F=1}^{6} \mathrm{DiffFlux_F}\right\}_{n+1} + \left\{\sum_{F=1}^{6} \mathrm{AdvFlux_F}\right\}_{n} - P_{n,Joule} - P_{n,Chem} - P_{n,Other} = F({}^kH_{n+1}) \tag{4.29}$$

The advection term (sum) has a semi-implicit form. A single term of the sum is written in more detail in Equation 4.30. The face velocity $\mathbf{u}_{F,n+1}$ is computed from converged cell-centered velocity values at the current time step ($t = t_{n+1}$), while the specific enthalpy is taken at $t = t_n$. Here, F is a cell face index which ranges from 1 to 6 on a hexahedral mesh. Non-liquid materials are assumed to be immobile in the code and thus do not contribute to enthalpy advection. In Equation 4.31, the explicit and semi-implicit advection terms are grouped into a quantity P_* which is computed once at the beginning of the time step. P_* is further used to set the first non-linear iterate of the Newton iteration at the new time level $n + 1$ (Equation 4.32).

$$\left\{ \mathrm{AdvFlux_F} \right\}_n = \mathbf{u}_{F,n+1} \hat{\mathbf{n}}_F A_F h_{liq,n}, \tag{4.30}$$

$$P_* = \left\{ \sum_{F=1}^{6} \mathrm{AdvFlux_k} \right\}_n + P_{n,Joule} + P_{n,Chem} + P_{n,Other} \tag{4.31}$$

$$^1H_{n+1} = P_* \delta t + H_n \tag{4.32}$$

The two remaining terms on the left hand side of Equation 4.29 are the flux term (sum) and the time derivative of H. The latter is given in Equation 4.33 and results in a second order time derivative approximation ($o(\delta t^2)$) at the half-time level ($t = t_n + 0.5\delta t$). For a locally orthogonal mesh, a single term in the total diffusive flux sum is written in Equation 4.34. Here, R_F is the vector connecting cell centers, A_F is the face area, k_F is face conductivity and $\hat{\mathbf{n}}_F$ is unit face normal. For a non-orthogonal mesh, the flux operator must be different. Truchas provides two types of flux operators to use for non-orthogonal meshes for heat transfer. See Appendix A and Appendix B for details. By default, the discretization for orthogonal meshes is fully implicit. This corresponds to $\theta = 1$ in Equation 4.35 and Equation 4.36. However, θ is a user defined quantity and is specified as $Conduction\_implicitness$ variable in the NUMERICS namelist. The stability restrictions on the diffusion term are more severe than those on the advection or source terms. The source terms currently do not depend explicitly on the enthalpy, or the temperature field. Therefore a fully implicit discretization is a sensible choice for the diffusion term.

$$\left\{ \frac{\partial H}{\partial t} \right\}_{n+1} = \frac{^kH_{n+1} - H_n}{\delta t} \tag{4.33}$$

$$\left\{ \mathrm{DiffFlux_F} \right\}_{n+1} = -\mathbf{R}_F \hat{\mathbf{n}}_F A_F k_F \frac{T(^kh_{n+1},\ h_n,\ \theta) - T(^kh_{n+1,F},\ h_{n,F},\ \theta)}{R_F^2} \tag{4.34}$$

For pure materials or alloys undergoing non-isothermal transformations, the cell-centered temperatures are evaluated by inverting a $h(T)$ thermodynamic relationship. For temperature dependent properties and in the phase change region of a binary alloy such an inversion also requires taking into account the volume fractions given by the solidification model and the concentrations given by the phase diagram. This is

achieved via a local cell-by-cell Newton-Raphson iteration (see the PHYSICS NAMELIST section for the $h(T)$ specification used in the code).

$$T({}^{k}h_{n+1},\ h_n,\ \theta) = \theta T({}^{k}h_{n+1}) + (1-\theta)T(h_n) \tag{4.35}$$

$$T({}^{k}h_{n+1,F},\ h_{n,F},\ \theta) = \theta T({}^{k}h_{n+1,F}) + (1-\theta)T(h_{n,F}) \tag{4.36}$$

4.2.2 Preconditioninig

The linear solution in Truchas requires preconditioning of the locally linearized non-linear system of equations. The Jacobian based linearization, and the details of the conjugate gradient and general minimum residual (GMRES) algorithms are described in detail in the NON-LINEAR SOLUTION chapter and Appendix C. A preconditionioning strategy adopted elsewhere in the code uses an approximation to Jacobian for improving the condition of the local (in time) linear system. The preconditioner is formed once at the beginning of each time step and the matrix formed in the code initially coincides with the Jacobian on orthogonal grids (the true Jacobian changes with every non-linear iteration). Therefore, the preconditioner can be written as follows:

$$\mathbf{P}_{n+1} \equiv \left.\frac{\partial \mathbf{F}(\mathbf{H})}{\partial \mathbf{H}}\right|_{{}^{1}\mathbf{H}_{n+1}}, \qquad P_{ij} = \frac{\partial F(H_i)}{\partial H_j} \tag{4.37}$$

Here, i and j are cell indexes. The preconditioner is sparse. On a hexahedral mesh and with a seven point diffusive flux operator the preconditioner has at most seven non-zero elements in every row. An off-diagonal and a diagonal elements of the preconditioner are formed in Equation 4.39 and Equation 4.38. In the current state of the code the form of preconditioner is the same for an orthogonal and a non-orthogonal mesh.

$$i = j, \quad P_{ii} = \delta t^{-1} - \theta \frac{\partial T_i}{\partial h_i} \frac{1}{Vol_i} \sum_{F=1}^{6} \frac{\mathbf{R}_{iF}\hat{\mathbf{n}}_{iF} A_{iF} k_{iF}}{R_{iF}^2} \tag{4.38}$$

$$i \neq j, \quad P_{ij} = \theta \frac{\partial T_j}{\partial h_j} \frac{1}{Vol_j} \frac{\mathbf{R}_{ij}\hat{\mathbf{n}}_{ij} A_{ij} k_{ij}}{R_{ij}^2} \tag{4.39}$$

Where $\mathrm{Vol}_i, \mathrm{Vol}_j$ are cell volumes and the $\frac{1}{C_p}$ factor is evaluated using the appropriate $h(T)$ relationship in every cell, taking into account possible multi-material (multi-phase) composition. In cells undergoing phase change, the C_p is replaced by the effective heat capacity ${}^{\text{eff}}C_p$, which incorporates the incremental release

of latent heat with an incremental change in the solid/liquid fraction. For example for a cell composed of solid with solid fraction f^S and liquid of liquid fraction $1 - f^S$ the effective heat capacity would read:

$$^{eff}C_p = C_{p,liq} + \frac{df_{sol}}{dT}(h_{sol}(T) - h_{liq}(T)) + f_{sol}(C_{p,sol} - C_{p,liq}) \tag{4.40}$$

The solid fraction increments with temperature are formed analytically using one of the currently available phase change models such as isothermal and non-isothermal transformations and binary alloy specific Scheil, Lever and Clyne and Kurz models. For an isothermal transformation the efffective heat capacity is set at a large finite value, while its inverse is essntially zero resulting in near zero off-diagonal terms and a diagonal term $P_{ii} = \delta t^{-1}$ for the cells undergoing phase transformation.

4.2.3 Heat Sources/Sinks

4.2.3.1 External Heat Source

All heat sources defined with this namelist are "volumetric heat sources," i.e. the amount of energy deposited in a cell over a given time step has the units of energy/volume. The integral of the heat source over the domain divided by the incremental time step gives the total power defined by the heat source constant. (Note that , in the case of the moving Gaussian below, if the heat source is defined partially outside of the domain this will not be the case.) The two cases below are mutually exclusive, such that only one or the other can be specified in the input file, and not both.

- **Moving Gaussian:**
  This type of source is distributed throughout some portion of the domain weighted by a Gaussian distribution, i.e. the change in energy, ΔE, in a cell due to the heat source over a time step Δt is given by,

$$\Delta E = \frac{A}{\pi^3 r_1 r_2 r_3} \exp\left(-\left(\frac{x_1}{r_1}\right)^2 - \left(\frac{x_2}{r_2}\right)^2 - \left(\frac{x_3}{r_3}\right)^2\right) \Delta t\, V \ , \tag{4.41}$$

  where (r_1, r_2, r_3) represents the radii of the heat source, (x_1, x_2, x_3) is the vector from the cell center to the centroid of the heat source, V is the cell volume and A is the heat source constant. The total amount of energy, ΔE_Ω over a time step, Δt, is determined by the equation

$$\Delta E_\Omega = \int_\Omega \frac{A}{\pi^3 r_1 r_2 r_3} \exp\left(-\left(\frac{x_1}{r_1}\right)^2 - \left(\frac{x_2}{r_2}\right)^2 - \left(\frac{x_3}{r_3}\right)^2\right) \Delta t\, dV \ , \tag{4.42}$$

  where Ω represents the domain. Note that $\Delta E_\Omega = A\Delta t$.

- **Picewise Constant:**
  This type of source is the sum of scaled characteristic functions of mutually disjoint subdomains Ω_i, $i = 1, \ldots, M$ of the domain Ω, such that

$$q(x, y, z) = \sum_{i=1}^{M} A_i \chi_{\Omega_i}(x, y, z), \tag{4.43}$$

  where A_i is the value of the volumetric heat source in subdomain Ω_i. The change in energy, ΔE in a cell due to the heat source over a time step Δt is given by

$$\Delta E = q \Delta t V, \tag{4.44}$$

  where V is the cell volume. In other words, if the cell is contained in Ω_i, then we have

$$\Delta E = A_i \Delta t V. \tag{4.45}$$

  The total amount of energy, ΔE_Ω, over a time step is determined by the equation

$$\Delta E_\Omega = \int_\Omega q \Delta t dV = \sum_{i=1}^{M} A_i \int_{\Omega_i} dV. \tag{4.46}$$

  We note that the special case, where $M = 1$ and $\Omega_1 = \Omega$ is possible. This is the case where a constant volumetric heat source is specified over the entire domain, $q(x, y, z) = A$.

Chapter 5

Chemical Reactions

The following chapter presents the chemistry capabilities incorporated into TRUCHAS. The chemistry section is responsible for calculating an exothermic heat source to be employed as a source term in the solution of the heat transfer component.

5.1 Physics

Chemical reactions in TRUCHAS are modeled as auto-catalytic reactions of the kind

$$A \rightarrow B[+C...] + \Delta h \tag{5.1}$$

We can write the time evolution equation of $C(t)$, the concentration of A, as:

$$\begin{aligned} \frac{\partial}{\partial t}C(t) &= (k_1 + k_2 * C(t)^m)(C_{max} - C(t))^n \\ k_1 &= k_1^o * e^{-E_1^a/RT} \\ k_2 &= k_2^o * e^{-E_2^a/RT} \end{aligned} \tag{5.2}$$

C is concentration of the product, t is time, k_1^o and k_2^o the reaction constants for the two stages and are both a function of temperature, T. The order of the reaction is given by $n + m$. R is the gas constant. It is further assumed that the heat of reaction, H_{tot} is evenly distributed as the reaction progresses. i.e.

$$\frac{\partial H}{\partial t} = H_{tot}\frac{\partial C}{\partial t} \tag{5.3}$$

5.1.1 Assumptions

It is assumed that only the forward reaction takes place. There is currently no mechanism in place to allow for reversible reactions other than to specify two chemical reactions with reactants and products switched and (possibly) new rate constants.

NOTE: R is currently assumed to be 8.314 J/mol^oK which limits us to using MKS units. Consequently the temperature must be the absolute temperature (^oK), the activatione energies should be in consistent units (J), and the the time and rates k_1, k_2, should be consistent.

5.1.2 Interaction With Other Physics

The reaction propagation is calculated in each of the cells explicitly at the beginning of the heat transfer solution and added to the enthalpy solution as a source term.

5.2 Algorithms

To allow for general reactions where we might not know the solution of the reaction equation, but rather only the differential equation governing the reaction, we solve the reaction as a fractional change in concentration during the time step Δt. This gives us:

$$\begin{aligned} \Delta C &= \Delta t \left[(k_1 + k_2 * C(t)^m)(C_{max} - C(t))^n \right] \\ k_1 &= k_1^o * e^{-E_1^a/RT} \\ k_2 &= k_2^o * e^{-E_2^a/RT} \end{aligned} \tag{5.4}$$

The above approach has the advantage of not needing the concentration history of the cells where the reaction takes place. However, it is prone to errors if the time step is large or if the reaction occurs very fast relative to the time step. Consequently for future models that may be added, where possible we will use the solution to the differential equation rather than the differential equation itself to propagate the reaction.

Once we have ΔC, we can then calculate the change in enthalpy, ΔH, associated with this concentration change.

This approach will not allow for chemical changes and phase changes to occur to the *same* material in any given time step, although different materials can undergo phase changes and chemical reactions, or the reactants can be formed as a result of a phase change from some other material.

Chapter 6

Solid Mechanics

The solid mechanics capability in TRUCHAS is described in this chapter. The current release can calculate displacements, elastic stresses and both elastic and plastic strains for an isotropic material, including stresses and deformations caused by temperature changes and gradients. The volume changes associated with solid state phase changes can also be included in the solution. A variety of traction and displacement boundary conditions can be specified, and this release includes sliding interfaces and contact, restricted to "small" displacements. The model for plastic flow uses a flow stress that depends on strain rate and temperature with no work hardening. Other material behavior such as porosity formation may be added in future versions of the software.

6.1 Notation

A	Control volume face area
$\mathbf{A}$	linear elastic operator
a_i, b_i, c_i, d_i	coefficients for tetrahedral linear interpolation
$\mathbf{b}$, b_i	body force components
d	displacement boundary condition value (scalar)
E	Young's modulus
e^{el}	elastic strain tensor
e^{pc}	phase change strain tensor
$\bar{e}^{pl}$	effective plastic strain (scalar)
e^{pl}	plastic strain tensor
e^{th}	thermal strain tensor
e^{tot}	total strain tensor
G	second Lame′ constant

$\mathbf{J}$	Jacobian matrix
N_i	Shape functions for a cell or element
$\hat{n}$, $\hat{n}_i$	unit normal vector and components
$\mathbf{P}$	preconditioning matrix
r, s, t	logical coordinates for element shape functions
$\mathbf{r}$,r_i	right hand side vector for the linear elastic system
T	temperature
T_{ref}	stress reference temperature
t	time
$\mathbf{u}$, u_i, $[u, v, w]$	displacement vector or components
x_i or $[x, y, z]$	global coordinates
α	linear coefficient of thermal expansion
δ_{ij}	Kronecker delta
λ	first Lame′ constant
ν	Poisson's ratio
ϕ	a displacement component
ρ	current density of the material (not accounting for thermal expansion)
ρ_0	initial density of the material
σ	Cauchy stress tensor
$\bar{\sigma}$	Effective stress (second invariant of σ, scalar)
σ'	deviatoric stress tensor
σ^{th}	thermal stress tensor
τ_i	traction components
τ_n	traction component normal to the surface
θ	temperature difference relative to a stress-free temperature

More notation specific to the mechanical threshold stress (MTS) model and the contact algorithm are listed in Section 6.2.5 and Section 6.2.3. Unless otherwise indicated, repeated indices denote summation, i.e. $e_{kk} \equiv e_{11} + e_{22} + e_{33}$.

6.2 Physics

6.2.1 Assumptions

The current solid mechanics implementation is an application of linear thermoelastic continuum mechanics with small strain viscoplastic flow. It is assumed that the solid material is a continuum, and the discretized strain field satisfies compatibility. The current formulation uses infinitesimal strains, and is therefore accurate only for small strains and rotations. The material behavior is assumed to be linear elastic with isotropic

J_2 plasticity. The deformation is assumed to be quasi-static. Body forces and isotropic dilatation due to phase change can be optionally included in the current implementation.

6.2.2 Equations

The basic conservation law to be satisfied is that of equilibrium (σ_{ij} are stress components, x_j are coordinates, b_i are body force components):

$$\frac{\partial \sigma_{ij}}{\partial x_j} + b_i = 0 \tag{6.1}$$

For isotropic elasticity:

$$\sigma_{ij} = \lambda e^{el}_{kk} \delta_{ij} + 2G e^{el}_{ij} \tag{6.2}$$

where λ and G are the first and second Lamé constants, respectively.

The strain tensor e^{el}_{ij} is the elastic strain defined by decomposing the total strain into elastic, thermal, plastic and phase change portions.

$$e^{tot}_{ij} = e^{el}_{ij} + e^{th}_{ij} + e^{pl}_{ij} + e^{pc}_{ij} \tag{6.3}$$

The thermal strain is defined as

$$e^{th}_{ij} = \alpha \delta_{ij} \theta \tag{6.4}$$

where α is the coefficient of thermal expansion, and θ is the temperature difference relative to a stress-free reference temperature. For the case of a temperature dependent coefficient of thermal expansion, Equation 6.4 becomes

$$e^{th}_{ij} = \int_{T_{ref}}^{T} \alpha(T) \delta_{ij} \, \mathrm{d}T \tag{6.5}$$

The phase change strain is also assumed to be an isotropic expansion or contraction calculated from the density change associated with the phase change. Assuming small strains, we use:

$$e^{pc}_{ij} = \left[\left(\frac{\rho_0}{\rho} \right)^{\frac{1}{3}} - 1 \right] \delta_{ij} \tag{6.6}$$

The total strain is the infinitesimal strain tensor defined in terms of the displacement gradient:

$$e^{tot}_{ij} = \frac{1}{2} \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) \tag{6.7}$$

where u_i are material displacement components. The displacement field is approximated by a finite volume discretization, which is described in Section 6.3. The displacements at cell vertices are the primary solution variables.

For isotropic thermo-elasticity with phase change, Equation 6.2, Equation 6.3, Equation 6.5 and Equation 6.6 give

$$\sigma_{ij} = \lambda e_{kk}^{tot}\delta_{ij} + 2Ge_{ij}^{tot} - (3\lambda + 2G)\left(\int_{T_{ref}}^{T} \alpha(T)\,\mathrm{d}T + \left[\left(\frac{\rho_0}{\rho}\right)^{\frac{1}{3}} - 1\right]\right)\delta_{ij} \tag{6.8}$$

The plastic strain is calculated from the users choice of viscoplastic model of the form

$$\frac{\mathrm{d}\bar{e}^{pl}}{\mathrm{d}t} = \mathrm{f}(\bar{\sigma}, T) \tag{6.9}$$

where $\bar{e}^{pl}$ is the effective plastic strain and $\bar{\sigma}$ is the second invariant of the stress tensor (J_2), or effective stress. The specific viscoplastic models available are described in Section 6.2.5. The effective plastic strain $\bar{e}^{pl}$ and effective stress $\bar{\sigma}$ are scalars, and the components of the plastic strain increment Δe_{ij}^{pl} are calculated from the Prandtl-Reuss equations

$$\Delta e_{ij}^{pl} = \frac{3}{2}\frac{\Delta\bar{e}^{pl}}{\bar{\sigma}}\sigma'_{ij} \tag{6.10}$$

where σ'_{ij} is the deviatoric stress and $\Delta\bar{e}^{pl}$ is the increment in effective plastic strain obtained by integrating Equation 6.9 over a time step.

The deviatoric stress is the elastic stress tensor minus the hydrostatic component.

$$\sigma'_{ij} = \sigma_{ij} - \delta_{ij}(\sigma_{kk}/3) \tag{6.11}$$

The plastic strain components are proportional to the deviatoric stress components, so the volumetric plastic strain (e_{kk}^{pl}) is zero. Including plastic strains in the stress calculation gives

$$\sigma_{ij} = \lambda e_{kk}^{tot}\delta_{ij} + 2Ge_{ij}^{tot} - (3\lambda + 2G)\left(\int_{T_{ref}}^{T} \alpha(T)dT + \left[\left(\frac{\rho_0}{\rho}\right)^{\frac{1}{3}} - 1\right]\right)\delta_{ij} - 2Ge_{ij}^{pl} \tag{6.12}$$

6.2.3 Boundary and Initial Conditions

6.2.3.1 Notation

$\mathbf{u}$	-	the displacement vector for the entire domain
$\hat{n}$	-	a unit vector normal to and pointing away from the surface
$\vec{u}_j$	-	the displacement vector (in ndim dimensions) for node j
$\vec{f}_j$	-	the force vector at node j that is a function of the displacement vector $\mathbf{u}$ and source terms
Λ	-	a contact function for a gap interface that is equal to 1 if the surfaces are in contact or penetrated and equal to 0 if the surfaces are separated.

6.2.3.2 Boundary Conditions

Boundary conditions may be specified in terms of displacements u_i or surface tractions. Since the primary solution variables are displacements, specifying displacements as Dirichlet conditions is straight forward. Tractions τ_j are defined as a force per unit area on a surface or interface, and are related to the bulk stress:

$$\tau_j = \sigma_{ij} \cdot \hat{n}_i \tag{6.13}$$

where τ_j are the traction components and $\hat{n}_i$ are the surface normal components. The algorithm used in TRUCHAS allows surface tractions to be specified directly on control volume faces. Displacements and tractions can be specified in either global Cartesian coordinates or in the direction normal to the surface or interface.

If a displacement normal to the surface is specified, the constraint equation is

$$\vec{u} \cdot \hat{n} = d \tag{6.14}$$

where $\mathbf{u}$ is the displacement vector, $\mathbf{n}$ is the unit normal vector at the node where the constraint is applied, and d is the scalar displacement value with a positive value in the direction pointing outward from the body.

Since the equilibrium equations for a node are for a force vector in three dimensions, the general case requires that the three equations for a node be decomposed into components normal and tangential to the surface. The equilibrium equations can be expressed as

$$\vec{f}_j = 0 \tag{6.15}$$

The displacement constraint can be applied by taking the component of the vector orthogonal to the specified displacement and adding the portion satisfying Equation 6.14:

$$([I] - [\hat{n}\hat{n}^T])\vec{f}_j - c[\hat{n}\hat{n}^T](\vec{u}_j - d\hat{n}) = 0 \tag{6.16}$$

where $[\hat{n}\hat{n}^T]$ is the 3×3 orthogonal projection matrix onto the normal direction, and $[I] - [\hat{n}\hat{n}^T]$ the orthogonal projection onto the plane normal to $\hat{n}$.

If a traction boundary condition normal to the surface is specified, the traction components in Cartesian coordinates are given by

$$\tau_i = \tau_n \hat{n}_i \tag{6.17}$$

where τ_i is the traction component in Cartesian coordinates, τ_n is the specified normal traction and $\hat{n}_i$ is the the i component of the unit normal vector. If τ_n is positive, the resulting force is in the direction pointing outward from the body.

6.2.3.3 Sliding Interfaces and Contact

Small displacement sliding interfaces can be specified, with or without a contact algorithm. The interface is defined with gap elements, that are currently constructed by duplicating mesh nodes and element faces on a surface and constructing elements of zero thickness by connecting the coincident faces and nodes in the mesh definition. In the future, elements of finite thickness may be designated as gap elements. The gap elements are currently only used to provide connectivity information to the sliding and contact algorithms, facilitating the parallel implementation.

Sliding interfaces specified without contact (designated "normal constraint" interfaces) allow coincident nodes across an interface to move relative to each other tangential to the surface but not normal to the surface. This is implemented by treating the nodes on the interface as if they are on a free surface and adding constraints that are dependent on the displacement vector of the coincident node on the other side of the interface. The constraints must satisfy two conditions: (1) the relative normal displacement is zero. For coincident nodes j and k on either side of the interface, and unit normal vector $\hat{n}$:

$$(\vec{u}_k - \vec{u}_j) \cdot \hat{n} = 0 \tag{6.18}$$

or in vector form

$$[\hat{n}\hat{n}^T](\vec{u}_k - \vec{u}_j) = 0 \tag{6.19}$$

and (2) the sum of the forces normal to the interface on the two nodes must be zero:

$$[\hat{n}\hat{n}^T](\vec{f}_j + \vec{f}_k) = 0 \tag{6.20}$$

To implement the sliding constraint, the equations for the force vectors acting on nodes j and k can be modified to be (for $\hat{n}$ pointing away from node j):

$$([I] - [\hat{n}\hat{n}^T])\vec{f}_j + [\hat{n}\hat{n}^T](\vec{f}_j + \vec{f}_k) + c[\hat{n}\hat{n}^T](\vec{u}_k - \vec{u}_j) = 0 \tag{6.21}$$

$$([I] - [\hat{n}\hat{n}^T])\vec{f}_k + [\hat{n}\hat{n}^T](\vec{f}_j + \vec{f}_k) + c[\hat{n}\hat{n}^T](\vec{u}_j - \vec{u}_k) = 0 \tag{6.22}$$

or

$$\vec{f}_j + [\hat{n}\hat{n}^T](\vec{f}_k + c(\vec{u}_k - \vec{u}_j)) = 0 \tag{6.23}$$
$$\vec{f}_k + [\hat{n}\hat{n}^T](\vec{f}_j + c(\vec{u}_j - \vec{u}_k)) = 0 \tag{6.24}$$

The nodal equations and constraints are in the form of the equation for a node on a free surface plus the force applied by the node across the interface plus a term that penalizes penetration or separation of the two nodes normal to the interface. The constant c is chosen to scale the penalty function appropriately.

Equation 6.24 can be modified with a contact function Λ to make the normal constraint non-symmetric, penalizing penetration but allowing the nodes to separate. Λ is equal to 1 if the surfaces are in contact or penetrated and equal to 0 if the surfaces are separated. Evaluating Λ at a node is an important implementation issue.

$$\vec{f}_j + \Lambda[\hat{n}\hat{n}^T](\vec{f}_k + c(\vec{u}_k - \vec{u}_j)) = 0 \tag{6.25}$$
$$\vec{f}_k + \Lambda[\hat{n}\hat{n}^T](\vec{f}_j + c(\vec{u}_j - \vec{u}_k)) = 0 \tag{6.26}$$

6.2.3.4 Initial Conditions

The temperature field is required for the thermoelastic solution, and initial temperatures that are specified in the input file or overwrite routine are used as initial conditions. A stress-free reference temperature must be specified for each material in the problem.

6.2.4 Interaction with Other Physics

The solid mechanics solution depends on the temperature field and volume fractions of the various solid phases. The volume fractions and temperature field at the beginning and end of the time step are used to calculate the thermal strain increments and plastic strain increments. Calculation of the phase change strain requires the change in volume from the enthalpy calculation. None of the other physics capabilities are dependent on the stresses, strains and displacements in this release.

6.2.5 Material Properties

6.2.5.1 Linear Elasticity

The isotropic thermoelastic model requires two elastic constants and a linear coefficient of thermal expansion. These properties are generally temperature dependent, and can be specified as functions of temperature in the input. Currently the elastic constants are specified as the Lamé constants λ and G, as defined in Equation 6.8. The relationship between Lamé constants and Young's modulus E and Poisson's ratio ν are given here:

$$\lambda = \frac{E\nu}{(1+\nu)(1-2\nu)} \tag{6.27}$$

$$G = \frac{E}{2(1+\nu)} \tag{6.28}$$

6.2.5.2 MTS Viscoplastic Model

The mechanical threshold stress (MTS) model [24] was developed to model plastic deformation of metals based on thermally activated deformation mechanisms. Although this model may not be the most appropriate for small strains and high temperatures, data for a number of materials of interest to the TELLURIDEproject have been fitted to this model.

If we ignore any terms related to work hardening, the equations for the MTS model with a strain rate and temperature dependent yield strength are as follows:

$$\frac{\sigma}{\mu} = \frac{\sigma_a}{\mu} + S_i(\dot{\epsilon}, T)\frac{\hat{\sigma}_i}{\mu_0} \tag{6.29}$$

$$\mu = \mu_0 - \frac{D}{\exp\left(\frac{T_0}{T}\right) - 1} \tag{6.30}$$

$$S_i(\dot{\epsilon}, T) = \left\{1 - \left[\frac{kT}{\mu b^3 g_{0i}} \ln\left(\frac{\dot{\epsilon}_{0i}}{\dot{\epsilon}}\right)\right]^{\frac{1}{q_i}}\right\}^{\frac{1}{p_i}} \tag{6.31}$$

The algorithm in TRUCHAS requires $\dot{\epsilon}$ as a function of T and σ. Equation 6.29 and Equation 6.31 can be solved for $\dot{\epsilon}$ to give

$$\dot{\epsilon} = \frac{\dot{\epsilon}_{0i}}{\exp\left\{\left[1 - \left(\frac{\mu_0}{\mu\hat{\sigma}_i}(\sigma - \sigma_a)\right)^{p_i}\right]^{q_i} \frac{\mu b^3 g_{0i}}{kT}\right\}} \tag{6.32}$$

The terms in the equations and corresponding TRUCHAS input variables are as follows:

Variable	Description	Input Variable
σ	Effective flow stress used in Equation 6.9	not an input variable
$\dot{\epsilon}$	Effective plastic strain rate used in Equation 6.9	not an input variable
μ	Temperature dependent shear modulus	not an input variable
T	Temperature	not an input variable
μ_0	Reference shear modulus	MTS_mu_0
σ_a	Athermal stress term	MTS_sig_a
$\hat{\sigma_a}$	Stress term for thermally activated yield stress	MTS_sig_i
D	Constant for temperature dependent shear modulus	MTS_d
T_0	Reference temperature for shear modulus	MTS_temp_0
k	Boltzmann's constant in appropriate units	MTS_k
b	Burger's vector magnitude	MTS_b
g_{0i}	Dimensionless constant	MTS_g_0i
$\dot{\epsilon}_{0i}$	Reference strain rate	MTS_edot_0i
p_i	Dimensionless constant	MTS_p_i
q_i	Dimensionless constant	MTS_q_i

6.2.5.3 Power Law Viscoplastic Model

The MTS model is not generally valid at high temperatures (relative to the melting point) and low strain rates. Accurate data for metals in this regime are often not available. A simple power law model is available to fit or estimate the plastic behavior in this regime. The relation between strain rate, stress and temperature is of the form

$$\dot{\epsilon} = A\bar{\sigma}^n \exp \frac{-Q}{RT} \tag{6.33}$$

where A, n and Q are material parameters.

6.3 Algorithms

6.3.1 Discretization

The discretization method used to solve Equation 6.8 or Equation 6.12 is based on a node-centered control volume discretization [4, 5]. This algorithm was chosen because it allows the efficient use of the existing

mesh data structures and parallel gather-scatter routines. Control volumes are constructed for each node or cell vertex using data from the mesh for fluid flow and heat transfer. Each cell is decomposed into sub-volumes with faces defined by connecting cell centroids, face centroids and edge midpoints, as shown in Figure 6.1.

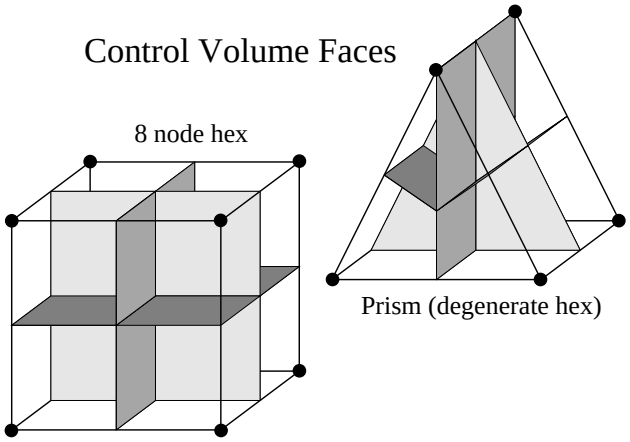


Figure 6.1: Faces defining control sub-volumes for a hex cell and a degenerate prism cell.

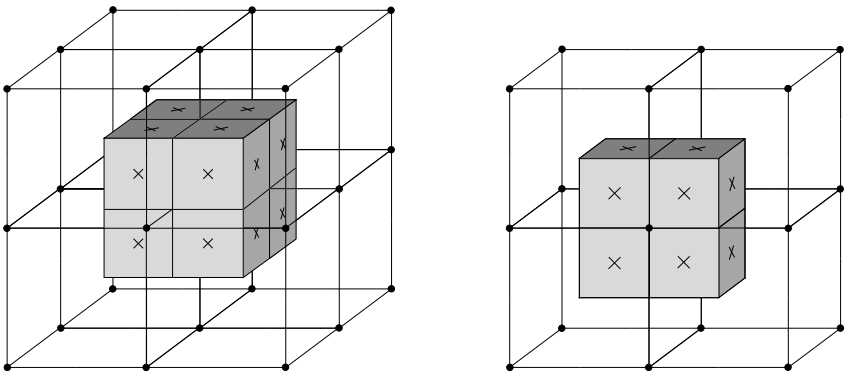


Figure 6.2: Faces defining control volumes for a node surrounded by hex cells and a node on the surface of a regular hex mesh.

Combining the faces of the sub-volumes from all cells that share a given node gives the surface of a control volume for that node. Control volumes for nodes that are on the surface of the mesh are enclosed by dividing the external cell face into smaller faces bounded by the internal control volume faces and cell edges. Examples for a structured hex mesh are shown in Figure 6.2.

The simplest derivation for the method used in TRUCHAS is to start with the equilibrium equation, Equation 6.1, and integrate over the control volume:

$$\int_V \frac{\partial \sigma_{ij}}{\partial x_j} \, \mathrm{d}v = -\int_V b_i \, \mathrm{d}v \tag{6.34}$$

The integral on the left hand side can be converted to a surface integral using the divergence theorem,

$$\int_V \frac{\partial \sigma_{ij}}{\partial x_j} \, \mathrm{d}v = \int_S \sigma_{ij} \cdot \hat{n}_j \, \mathrm{d}s \tag{6.35}$$

where the integral is now defined over the surface of the control volume.

Combining with Equation 6.12, moving the thermal stresses and phase change stresses to the right hand side gives

$$\int_S (\lambda e_{kk}^{tot} \delta_{ij} + 2G(e_{ij}^{tot} - e_{ij}^{pl})) \cdot \hat{n}_j \, \mathrm{d}s = \int_S \left[\left(\int_{T_{ref}}^{T} \alpha(T) \, \mathrm{d}T + \left[\left(\frac{\rho_0}{\rho} \right)^{\frac{1}{3}} - 1 \right] \right) \delta_{ij} \right] \cdot \hat{n}_j \, \mathrm{d}s - \int_V b_i \, \mathrm{d}v \tag{6.36}$$

for each control volume. Bailey and Cross [5] claim that Equation 6.36 can also be derived from a weighted residual method where the weight function is unity within the control volume and zero elsewhere.

6.3.2 Displacement Gradients

The total strain e_{ij}^{tot} is evaluated at the face centroids by standard finite element techniques for tri-linear hexahedral elements. The displacement field within an mesh cell is related to nodal displacements by

$$\phi(r,s,t) = \sum_{i=1}^{m} N_i(r,s,t)\phi_i \tag{6.37}$$

where $\phi(r,s,t)$ is a displacement component at logical coordinates r,s,t within a reference element, N_i are shape functions at those same coordinates, and ϕ_i are the corresponding displacements at the nodes. The global coordinates x,y,z are related to r,s,t through the Jacobian matrix

$$\mathbf{J} = \begin{pmatrix} \frac{\partial x}{\partial r} & \frac{\partial y}{\partial r} & \frac{\partial z}{\partial r} \\ \frac{\partial x}{\partial s} & \frac{\partial y}{\partial s} & \frac{\partial z}{\partial s} \\ \frac{\partial x}{\partial t} & \frac{\partial y}{\partial t} & \frac{\partial z}{\partial t} \end{pmatrix} \tag{6.38}$$

where

$$\frac{\partial x}{\partial r} = \sum_{i=1}^{m} \frac{\partial N_i}{\partial r} x_i \quad \text{etc.} \tag{6.39}$$

After inverting $\mathbf{J}$, displacement gradient components in global coordinates can be calculated:

$$\begin{bmatrix} \frac{\partial \phi}{\partial x} \\ \\ \frac{\partial \phi}{\partial y} \\ \\ \frac{\partial \phi}{\partial z} \end{bmatrix} = \mathbf{J^{-1}} \begin{bmatrix} \frac{\partial \phi}{\partial r} \\ \\ \frac{\partial \phi}{\partial s} \\ \\ \frac{\partial \phi}{\partial t} \end{bmatrix} \tag{6.40}$$

6.3.3 Solution Algorithm for Quasi-Static Stresses and Strains

The left hand side of Equation 6.36 is evaluated from the strains at the centroid of each control volume face and the normal at the face centroid using a single quadrature point. A linear thermo-elastic solution is computed on initialization if solid mechanics is active. The initial solution uses either the Newton-Krylov or accelerated inexact Newton nonlinear solver for the displacements.

The solution algorithm computes all stress and strain quantities at all integration points for all cells. These quantities are also recalculated at cell centroids for output purposes only.

6.3.3.1 Initialization

Since the current implementation is a small strain Lagrangian formulation, the Jacobians are only calculated once upon initialization. Initialization for the solid mechanics solution includes the following:

- Calculate and store the area A, centroid coordinates and normal vector $\hat{n}_i$ for each control volume face in each cell.
- Calculate and store the inverse of the Jacobian matrix, $\mathbf{J^{-1}}$ for each control volume face in each cell.
- Identify external cell faces that have traction boundary conditions specified, and calculate areas A and normal vectors for control volume faces on those cell faces.
- Identify nodes for displacement boundary conditions.
- If gap elements are present, identify pairs of nodes for interface and contact constraints. Also calculate and store the various normal and tangential vectors for the contact algorithm.

6.3.3.2 Initial Thermo-Elastic Solution

The solvers used in TRUCHAS do not require the formation of a stiffness matrix, except for preconditioning, as described in Section 6.3.4. The solution method for the initial linear thermo-elastic stresses, strains and displacements is as follows:

- Calculate the source terms that do not depend on the displacement vector in Equation 6.36, denoted as r:
  - For each cell, calculate the thermal stress contribution from the cell temperature:

$$\sigma_{ij}^{th}\delta_{ij} = (3\lambda + 2G)\left(\int_{T_{ref}}^{T} \alpha(T)\,\mathrm{d}T\right) \tag{6.41}$$

  - For each cell, calculate the phase change stress contribution from the volume change calculated as part of the enthalpy solution:

$$\sigma_{ij}^{pc}\delta_{ij} = (3\lambda + 2G)\left[\left(\frac{\rho_0}{\rho}\right)^{\frac{1}{3}} - 1\right] \tag{6.42}$$

  - Accumulate the right hand side surface integrals for each displacement component i for each node for each cell by summing over the number of control volume faces for the node (nfaces). Using values at the control volume face centroid (with no summation over index i):

$$r_i = \sum_{m=1}^{\text{nfaces}} (\sigma_{ii}^{th} + \sigma_{ii}^{pc})\hat{n}_i^m A^m \tag{6.43}$$

  - Add contributions from traction boundary conditions τ_i:

$$r_i = r_i + \sum_{m=1}^{\text{nfaces}} \tau_i^m A^m \tag{6.44}$$

  - Add gravitational body force terms if required

$$b_i = V_n \rho_n \vec{g}_i \tag{6.45}$$

    where V_n is the volume of the control volume for the node, ρ_n is the average density of the control volume for the node and $\vec{g}$ is the gravitational acceleration vector
  - Enforce displacement, sliding interface and contact boundary conditions described in Section 6.2.3.2 and Section 6.2.3.3. The detailed equations for the supported combinations of displacement, sliding and contact constraints are in Appendix K.

– Scatter the contributions to the right hand side for all cells to the nodes.

- Call the non-linear solver package to calculate displacements at the nodes. The non-linear residual routine is called by the solver routines. Although the initial solution is a linear elastic solution, if contact boundary conditions are specified, the problem is non-linear.

- Calculate the total strain field at the cell centroid and at each integration point in each cell from the displacements. (Equation 6.7).

- Subtract the thermal and phase change strains to get the elastic strain:

$$e_{ij}^{el} = e_{ij}^{tot} - \alpha\theta\delta_{ij} \tag{6.46}$$

- Calculate the elastic stress at the cell centroid and at each integration point in each cell from the displacements using Equation 6.2.

- Calculate the deviatoric stress using Equation 6.11.

- Calculate the effective stress at the cell centroid and at each integration point in each cell using

$$\bar{\sigma} = \left[\left((\sigma_{11}-\sigma_{22})^2 + (\sigma_{22}-\sigma_{33})^2 + (\sigma_{33}-\sigma_{11})^2 + 6(\sigma_{12}^2+\sigma_{13}^2+\sigma_{23}^2)\right)/2\right]^{\frac{1}{2}} \tag{6.47}$$

- Calculate and store the plastic strain rate at the cell centroid and at each integration point in each cell.

6.3.3.3 Non-Linear Thermo-Elastic-Viscoplastic Solution

After the initial linear thermoelastic solution, it is assumed that the Equation 6.36 is non-linear with material properties that may be temperature dependent. The solution procedure for each time step is as follows:

- Store the elastic stress, plastic strain and total strain components for each integration point for each cell from the initial solution or the previous time step.

- Calculate the deviatoric stress at each integration point using Equation 6.11.

- Calculate the effective stress using Equation 6.47.

- Calculate and store the portion of the residual that does not depend on the displacement field. This currently includes the thermal strain terms, phase change strain terms. These terms are also modified by the displacement, sliding and contact constraints.

- Call the non-linear solver to calculate the displacement field. The non-linear solver calls routines for the residual calculation and possibly a numerical approximation of the Jacobian matrix-vector product as described below. The stresses and strains at the integration points are updated automatically when the residual is computed for the convergence check.
- Calculate the total strain, plastic strain, elastic stress, and plastic strain rate at the cell centroids.

6.3.3.4 Residual Calculation:

The accelerated inexact Newton and Newton-Krylov methods both require the calculation of the solution residual for a given displacement vector. The residual is given by

$$\mathbf{F}(\mathbf{u}) = \frac{\partial \sigma_{ij}}{\partial x_i} + \mathbf{b} \tag{6.48}$$

where σ_{ij} is given by Equation 6.12. The residual is calculated by evaluating σ_{ij} at each integration point and integrating over the control volume as for the linear elastic solution.

$$F_j(\mathbf{u}) = \sum_{m=1}^{\text{nfaces}} \sigma_{ij}^m \hat{n}_i^m A^m + b_j \tag{6.49}$$

Note that σ_{ij} is calculated from the elastic strain, which requires calculating the total strain, thermal strain and plastic strain at each control volume face.

$$e_{ij}^{el} = e_{ij}^{tot} - e_{ij}^{th} - e_{ij}^{pl} - e_{ij}^{pc} \tag{6.50}$$

The total strain e^{tot} is calculated from the displacement field as described above, and the thermal and phase change strain terms are computed and stored once for each time step. The plastic strain is calculated by integrating the plastic strain rate at each integration point over the time step. The integration of the plastic strain is implicit, assuming a linear change in total strain over the time step and a constant straining direction equal to that at the beginning of the time step. If the plastic strain rate is small enough a midpoint method is used to calculate an average strain rate. Otherwise an ODE integrator (BDF2) is used to accurately integrate the plastic strain over the time step at an integration point.

6.3.3.5 Boundary Conditions

Traction boundary conditions are applied directly to the control volume faces on the boundary by substituting τ_j for $(\sigma_{ij}\hat{n}_i)$ in Equation 6.49.

Displacement boundary conditions in Cartesian coordinates are applied by replacing the equations for the node with Equation 6.16. Sliding and contact interface conditions are imposed according to Equation 6.24

and Equation 6.26. The details of the projections of the force vectors for displacement and interface boundary conditions are in Appendix K.

6.3.4 Preconditioning

For efficient convergence the nonlinear solvers require an approximation of the Jacobian matrix ($\frac{\partial \mathbf{F}}{\partial \mathbf{u}}$) for preconditioning. Currently TRUCHAS uses an approximation of the elastic stiffness matrix $\mathbf{A}$ for preconditioning of the linear and nonlinear solvers. The method for approximating $\mathbf{A}$ developed for TRUCHAS uses the same control volumes and surface integral as for the full operator $\mathbf{A}$, but the displacement gradients are approximated using only neighboring nodes connected by edges. Instead of using the full tri-linear interpolation functions and all eight nodes from a cell, three adjacent nodes from a single cell are used to construct a tetrahedron (Figure 6.3), where stresses are to be calculated for the surface of the volume surrounding the node labeled 1.

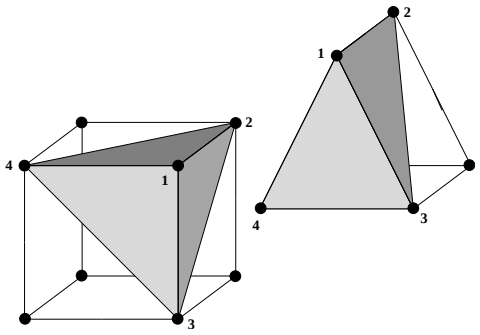


Figure 6.3: Definition of tetrahedra for displacement gradient approximation.

The displacement components (represented by u, v, w) are assumed to vary linearly over the tetrahedron:

$$\begin{aligned} u &= \sum_{i=1}^{4}(a_i + b_i x + c_i y + d_i z)u_i \\ v &= \sum_{i=1}^{4}(a_i + b_i x + c_i y + d_i z)v_i \\ w &= \sum_{i=1}^{4}(a_i + b_i x + c_i y + d_i z)w_i \end{aligned} \tag{6.51}$$

The displacement gradients are then constant over the element,

$$\begin{aligned}
\frac{\partial u}{\partial x} &= \sum_{i=1}^{4} b_i u_i \\
\frac{\partial u}{\partial y} &= \sum_{i=1}^{4} c_i u_i \\
&\vdots \\
\frac{\partial w}{\partial z} &= \sum_{i=1}^{4} d_i w_i
\end{aligned} \tag{6.52}$$

where (u_1, u_2, u_3, u_4) are the u displacements at nodes 1-4 in Figure 6.3, etc. The coefficients (a_i, b_i, c_i, d_i) are calculated from the coordinates at nodes 1-4, as described in various finite element references, such as [25]. The elastic equilibrium equation $\mathbf{Au} = \mathbf{b}$ can then be expanded using an approximation of $\mathbf{A}$ in terms of (a_i, b_i, c_i, d_i) and the control volume face areas and normals. A single stress value based on the tetrahedron strain gradients is used for all control volume faces associated with node 1 in that cell.

This procedure results in a matrix $\mathbf{P}$, an approximation to $\mathbf{A}$. For hexahedral meshes $\mathbf{P}$ usually has a much smaller bandwidth than $\mathbf{A}$. If the mesh is fixed, most of the work to calculate $\mathbf{P}$ (evaluation of a_i, b_i, c_i, d_i) is done once at the beginning of the simulation. If λ and G are temperature dependent or if material volume fractions in a cell change, $\mathbf{P}$ must be recalculated.

The linear solver package uses $\mathbf{P}$ for preconditioning of the linear system. Currently the only preconditioning methods for solid mechanics are symmetric successive over-relaxation (SSOR) and diagonal scaling. SSOR generally results in a reduction of the number of iterations by a factor of 3 or 4 compared to no preconditioning. The preconditioning routine itself is quite efficient, and the reduction in computation time is substantial.

For tetrahedral meshes $\mathbf{P}$ should be equal to $\mathbf{A}$. Computations on tetrahedral meshes can be quite efficient, but the accuracy of linear tetrahedra are quite poor compared to tri-linear hexes for the same number of nodes. It should also be noted that tetrahedral meshes generally have a much larger number of cells for a given number of nodes. Since the algorithms in TRUCHAS generally loop over cells for the construction of $\mathbf{A}$ and $\mathbf{P}$, the benefits of tet meshes are reduced.

Chapter 7

Electromagnetics

This chapter presents the electromagnetic (EM) modeling capabilities in TRUCHAS, and discusses the solution procedure used for EM field equations in detail. Although the core EM solver is reasonably general, its current application within TRUCHAS is narrowly focused on treating induction heating problems—problems where a workpiece is surrounded by an induction coil that generates a low-frequency magnetic field. This focus has guided the special manner EM is coupled to the other physics in TRUCHAS. While there are a number of limitations in this initial release, the current induction heating capability is still quite useful, and will be improved in future releases. It is also expected that the EM modeling capabilities will be expanded in the future to include other phenomena, such as magnetic stirring of fluids due to Lorentz forces.

7.1 Physics

7.1.1 Assumptions

The EM implementation is based on a direct solution of Maxwell's equations in the time domain. The material parameters—permittivity, permeability, and electrical conductivity—are assumed to be isotropic and linear in the fields. Imposed currents within the computational domain are not presently treated.

Other assumptions are:

Low frequency driving field: The frequency of the magnetic driving field is assumed to be low in the sense that its wavelength is large compared to the diameter of the computational domain. This is due to the way in which the driving fields are constructed. The displacement current term in Maxwell's equations, which is normally dropped in the low-frequency regime, is retained in this implementation.

Separation of time scales: The time scale associated with the EM fields, which is inversely proportional to the driving field frequency, is assumed to be much shorter than the time scale associated with the other physics, particularly heat conduction. It is this assumption which allows the special coupling used in this implementation. So while we require a low-frequency driving field, its frequency must not be too low.

External induction coil: Because imposed currents are not treated in the present implementation, it is not possible to directly model induction coils within the computational domain. Instead, the influence of the coil is captured through boundary conditions on the magnetic field. What is assumed is that there is sufficient free-space separation between the coil and workpiece, so that the boundary of the computational domain may be placed at an intermediate point far enough away from the workpiece so that the reaction fields induced in the workpiece are not unduly affected by the boundary, yet still excluding the coil itself.

Tetrahedral mesh: The current numerical procedure used to solve the EM field equations requires a tetrahedral mesh. Thus TRUCHAS now operates with a pair of meshes: a secondary tet mesh used for the EM calculations and the primary mesh used elsewhere in TRUCHAS. Quantities are interpolated from one mesh to the other as needed.

Fixed domain type: The computational domain is required to be one of a few specific types. This is due to the current inability to associate boundary conditions to specific portions of the boundary of the mesh used for the EM calculation. This limitation will be removed in a future release. This issue is discussed in more detail later in this section.

7.1.2 Equations

The evolution of general time-varying EM fields is governed by Maxwell's equations, which in SI units can be written as

$$\frac{\partial \vec{B}}{\partial t} = -\nabla \times \vec{E}, \qquad \text{(Faraday's law)}, \tag{7.1}$$

$$\frac{\partial \vec{D}}{\partial t} + \vec{J} = \nabla \times \vec{H}, \qquad \text{(Ampere-Maxwell law)}, \tag{7.2}$$

$$\nabla \cdot \vec{D} = \rho, \qquad \text{(Gauss' electric law)}, \tag{7.3}$$

$$\nabla \cdot \vec{B} = 0, \qquad \text{(Gauss' magnetic law)}. \tag{7.4}$$

Here $\vec{E}$ is the electric field intensity, $\vec{H}$ is the magnetic field intensity, $\vec{B}$ is the magnetic flux density, $\vec{D}$ is the electric flux density, $\vec{J}$ is the electric current density, and ρ is the electric charge density. In addition we have a continuity equation governing conservation of charge,

$$\frac{\partial \rho}{\partial t} + \nabla \cdot \vec{J} = 0. \tag{7.5}$$

The fields are connected by the constitutive relations

$$\vec{D} = \epsilon \vec{E}, \tag{7.6}$$

$$\vec{B} = \mu \vec{H}, \tag{7.7}$$

$$\vec{J} = \sigma \vec{E} + \vec{J}_{\text{src}}, \tag{7.8}$$

where the parameters ϵ, μ, and σ denote, respectively, the permittivity, permeability, and conductivity of the medium. Currently we consider only isotropic materials, where these parameters are scalars. The field $\vec{J}_{\text{src}}$ denotes an imposed current density. However, in this release we assume $\vec{J}_{\text{src}} \equiv 0$; this will be relaxed in a future version.

Finally, the Joule heat, q, which couples the electromagnetics to heat conduction, is simply computed as

$$q = \sigma \|\vec{E}\|^2. \tag{7.9}$$

This is a power density.

7.1.3 Boundary Conditions

Internally to TRUCHAS, either the tangential component of the electric field or the tangential component of the magnetic field may be specified on the boundary. Specifically, suppose the boundary is partitioned into two disjoint parts, Γ_1 and Γ_2, either one possibly empty. Then the boundary conditions are

$$\hat{n} \times \vec{E} = \hat{n} \times \vec{E}_b, \quad \text{on } \Gamma_1, \qquad \text{(Type 1)} \tag{7.10}$$

$$\hat{n} \times \vec{H} = \hat{n} \times \vec{H}_b, \quad \text{on } \Gamma_2, \qquad \text{(Type 2)} \tag{7.11}$$

where $\hat{n}$ denotes the outward normal to the boundary, and $\vec{E}_b(\mathbf{x}, t)$ and $\vec{H}_b(\mathbf{x}, t)$ are given boundary data.

Currently, however, there is no means for the user to associate boundary conditions to specific portions of the boundary of the secondary tetrahedral mesh used for the EM calculation. As a result the domain Ω is limited to three special types that are typical of basic induction heating problems:

- Full cylinder, $\Omega = \{(x, y, z) \mid x^2 + y^2 \leq r^2,\ z_1 \leq z \leq z_2\}$
- Half cylinder, $\Omega = \{(x, y, z) \mid x^2 + y^2 \leq r^2,\ x \geq 0,\ z_1 \leq z \leq z_2\}$
- Quarter cylinder, $\Omega = \{(x, y, z) \mid x^2 + y^2 \leq r^2,\ x, y \geq 0,\ z_1 \leq z \leq z_2\}$

The half and quarter cylinder domains are assumed to be associated with a full cylinder problem that possesses half and quarter symmetry, respectively. If present, the symmetry boundaries $\partial\Omega \cap \{x = 0\}$ and $\partial\Omega \cap \{y = 0\}$ are assigned to Γ_1 with $\vec{E}_b = \vec{0}$. This is consistent with the symmetry that allows a globally azimuthal electric field. The remaining portion of the boundary is assigned to Γ_2, and several pre-defined choices for $\vec{H}_b$ may be selected.

7.1.3.1 Magnetic driving fields.

The current choices for $\vec{H}_b$ all correspond to the magnetic field produced by a cylindrical coil, of some configuration, that carries a sinusoidally varying current. They may be expressed in the form

$$\vec{H}_b(\mathbf{x}, t) = I \sin(2\pi f t)\vec{h}(\mathbf{x}), \tag{7.12}$$

where I is the peak current per unit length, f is the linear frequency, and $\vec{h}(\mathbf{x})$ is a vector field that depends on the geometric configuration of the coil.

7.1.4 Interaction With Other Physics

The EM field solution is independent of all other physics, except temperature, and that only if the EM material parameters are temperature dependent. Heat conduction is coupled to EM through the Joule heat which serves as a volumetric heat source in the enthalpy equation. The coupling between the two physics is greatly simplified, however, by the fundamental assumption that the time scale associated with the EM fields (inversely proportional to the frequency of the magnetic driving field) is much shorter than the time scale associated with heat conduction. In this case, the EM field persists in a periodic steady-state equilibrium that continually adjusts to the slowly evolving temperature field. To find this steady state, it suffices to solve Maxwell's equation to the periodic steady state, starting from arbitrary initial conditions (zero fields, for example), while temporarily freezing all other physics. Finally, the rapid temporal fluctuations in the derived Joule heat is removed by averaging over a cycle, yielding the time-averaged heat source used in heat conduction.

7.1.5 Material Properties

The material parameters relevant to EM are the permittivity ϵ, permeability μ, and electrical conductivity σ. The first two are expressed in terms of their free-space values, $\epsilon = \epsilon_0\epsilon_r$ and $\mu = \mu_0\mu_r$, where ϵ_r and μ_r are the relative permittivity and permeability, respectively. The parameters ϵ_r, μ_r, and σ are specified in the material input and may be temperature dependent. The free-space parameters ϵ_0 and μ_0 are pre-assigned SI-unit values, but these may be overridden if necessary.

7.2 Algorithms

7.2.1 The Whitney Complex

Let $\mathcal{T}_h$ be a discretization of the domain Ω into a face-conforming tetrahedral mesh. Let $\mathcal{N}$, $\mathcal{E}$, $\mathcal{F}$, and $\mathcal{K}$ denote the sets of nodes, oriented edges and faces, and tetrahedral cells in the mesh. Each edge and face appears just once with a fixed but arbitrary orientation.

We define the Whitney family of finite element spaces $\mathcal{W}^0$, $\mathcal{W}^1$, $\mathcal{W}^2$, and $\mathcal{W}^3$ that are associated with the nodes, edges, faces, and cells of $\mathcal{T}_h$. Let $\phi_n(\mathbf{x})$ denote the familiar continuous, piecewise-linear, 'hat' function that equals 1 at node n, and 0 at all other nodes. Then for each oriented edge $\varepsilon = [m, n]$ define the vector function

$$\vec{w}_\varepsilon^{(1)}(\mathbf{x}) = \phi_m \nabla \phi_n - \phi_n \nabla \phi_m, \tag{7.13}$$

and for each oriented face $f = [l, m, n]$ define the vector function

$$\vec{w}_f^{(2)}(\mathbf{x}) = 2(\phi_l \nabla \phi_m \times \nabla \phi_n + \phi_m \nabla \phi_n \times \nabla \phi_l + \phi_n \nabla \phi_l \times \nabla \phi_m). \tag{7.14}$$

These functions have the following properties (see [26]):

- $\vec{w}_\varepsilon^{(1)}$ is tangentially continuous, and $\vec{w}_f^{(2)}$ is normally continuous across each face.
- The tangential component of $\vec{w}_\varepsilon^{(1)}$ is constant on each edge, its circulation along edge ε equals 1, and equals 0 for all other edges.
- The normal component of $\vec{w}_f^{(2)}$ is constant on each face, its flux across face f equals 1, and equals 0 for all other faces.
- The sets $\{\vec{w}_\varepsilon^{(1)}\}_{\varepsilon \in \mathcal{E}}$ and $\{\vec{w}_f^{(2)}\}_{f \in \mathcal{F}}$ are linearly independent.

We then let $\mathcal{W}^1 = \text{span}\{\vec{w}_\varepsilon^{(1)}\}$ and $\mathcal{W}^2 = \text{span}\{\vec{w}_f^{(2)}\}$. The degrees of freedom (DOF) for $\mathcal{W}^1$ are the circulations of a vector field along the oriented edges, and the DOF for for $\mathcal{W}^2$ are the fluxes of a vector field across the oriented faces. For completeness we also let $\mathcal{W}^0 = \text{span}\{\phi_n\}$, whose DOF are simply the nodal values of a scalar field, and let $\mathcal{W}^3$ denote the space of piecewise constant functions with respect to $\mathcal{T}_h$, whose DOF are the total masses of a scalar field on the cells. (We could just as well used the constant cell values as the DOF—it's just a matter of a change in basis.)

The functions in $\mathcal{W}^0$ are continuous, hence ∇u is defined for all $u \in \mathcal{W}^0$; that is, $\mathcal{W}^0$ is grad-conforming, $\mathcal{W}^0 \subset H(\Omega, \nabla)$. Similarly, it follows from the above properties that $\mathcal{W}^1$ is curl-conforming, $\mathcal{W}^1 \subset H(\Omega, \nabla\times)$, and $\mathcal{W}^2$ is div-conforming, $\mathcal{W}^2 \subset H(\Omega, \nabla\cdot)$. Moreover, it can be shown (non-trivial) that

$\nabla(\mathcal{W}^0) \subset \mathcal{W}^1$, $\nabla \times (\mathcal{W}^1) \subset \mathcal{W}^2$, and $\nabla \cdot (\mathcal{W}^2) \subset \mathcal{W}^3$. These key properties of the Whitney complex are summarized in the following diagram:

$$\mathcal{W}^0 \xrightarrow{\nabla} \mathcal{W}^1 \xrightarrow{\nabla\times} \mathcal{W}^2 \xrightarrow{\nabla\cdot} \mathcal{W}^3 \tag{7.15}$$

In particular it follows that $\nabla \times (\nabla u) \equiv 0$ for all u in the finite dimensional space $\mathcal{W}^0$, and $\nabla \cdot (\nabla \times \vec{v}) \equiv 0$ for all $\vec{v}$ in $\mathcal{W}^1$.

7.2.2 Spatial Discretization

Not all of the equations (7.1)–(7.5) are independent. Equation (7.4) follows from (7.1) provided that $\nabla \cdot \vec{B} = 0$ at the initial time. Equation (7.5) follows from (7.2) and (7.3). Moreover, since the charge density is of no direct interest, we ignore (7.3), other than to require that $\nabla \cdot \vec{D} = \rho$ at the initial time. These conditions on the initial fields may be trivially satisfied by considering a charge-free initial state that is free of fields, as we do here. What remains then are the first order curl equations (7.1) and (7.2), which after eliminating $\vec{D}$, $\vec{H}$, and $\vec{J}$, become

$$\frac{\partial \vec{B}}{\partial t} = -\nabla \times \vec{E}, \tag{7.16}$$

$$\epsilon \frac{\partial \vec{E}}{\partial t} + \sigma \vec{E} = \nabla \times \tfrac{1}{\mu} \vec{B}. \tag{7.17}$$

Let $\vec{B}_h \in \mathcal{W}^2$ and $\vec{E}_h \in \mathcal{W}^1$ be finite element approximants to $\vec{B}$ and $\vec{E}$, respectively. Because of the inclusion $\nabla \times (\mathcal{W}^1) \subset \mathcal{W}^2$, Faraday's law (7.16) may be discretized directly,

$$\frac{\partial \vec{B}_h}{\partial t} = -\nabla \times \vec{E}_h. \tag{7.18}$$

Ampere's law (7.17), however, must be interpreted weakly. The weak-form equation is

$$\int_\Omega \left(\epsilon \frac{\partial \vec{E}_h}{\partial t} + \sigma \vec{E}_h\right) \cdot \vec{w} = \int_\Omega \tfrac{1}{\mu} \vec{B}_h \cdot \nabla \times \vec{w} + \int_{\Gamma_2} \vec{w} \cdot \hat{n} \times \vec{H}_b, \quad \text{for all } \vec{w} \in \mathcal{W}^1. \tag{7.19}$$

The type 2 boundary condition (7.11) is imposed naturally through the boundary integral. The type 1 boundary condition (7.10) on the tangential component of $\vec{E}$ is also easily imposed on $\vec{E}_h$ whose DOF are the circulation of the electric field along the edges.

The discrete system (7.18) and (7.19) can be recast in matrix form. Suppose the edges and faces have been enumerated, $\mathcal{E} = \{\varepsilon_j\}_{j=1}^{N_\varepsilon}$ and $\mathcal{F} = \{f_j\}_{j=1}^{N_f}$, and write $\vec{E}_h$ and $\vec{B}_h$ in terms of their respective bases,

$$\vec{E}_h(\mathbf{x}, t) = \sum_{j=1}^{N_\varepsilon} e_j(t)\ \vec{w}^{(1)}_{\varepsilon_j}(\mathbf{x}), \tag{7.20}$$

$$\vec{B}_h(\mathbf{x}, t) = \sum_{j=1}^{N_f} b_j(t)\ \vec{w}^{(2)}_{f_j}(\mathbf{x}). \tag{7.21}$$

Set $\mathbf{e}(t) = (e_1, \ldots, e_{N_\varepsilon})$ and $\mathbf{b}(t) = (b_1, \ldots, b_{N_f})$. Then (7.18) and (7.19) can be written

$$\dot{\mathbf{b}} = -C\mathbf{e}, \tag{7.22}$$

$$M_1(\epsilon)\dot{\mathbf{e}} + M_1(\sigma)\mathbf{e} = C^T M_2(\mu^{-1})\mathbf{b} + \mathbf{g}(t), \tag{7.23}$$

where C is the matrix representation of the curl operator as a map from $\mathcal{W}^1$ to $\mathcal{W}^2$, $\mathbf{g}(t)$ stems from the boundary integral, and $M_1(\epsilon)$, $M_1(\sigma)$, and $M_2(\mu^{-1})$ are mass matrices defined by

$$M_k(\omega) = \left(\left(\int_\Omega \vec{w}_i^{(k)} \cdot \vec{w}_j^{(k)}\, \omega(\mathbf{x}) d\mathbf{x}\right)\right) \tag{7.24}$$

7.2.3 Time Discretization

We discretize the time derivatives in (7.22) and (7.23) using the trapezoid rule. Using superscripts to denote the time level, and a time step of Δt we obtain

$$\mathbf{b}^{n+1} - \mathbf{b}^n = -\tfrac{\Delta t}{2} C(\mathbf{e}^{n+1} + \mathbf{e}^n), \tag{7.25}$$

$$\begin{aligned} &M_1(\epsilon)(\mathbf{e}^{n+1} - \mathbf{e}^n) + \tfrac{\Delta t}{2} M_1(\sigma)(\mathbf{e}^{n+1} + \mathbf{e}^n) = \\ &\qquad \tfrac{\Delta t}{2} C^T M_2(\mu^{-1})(\mathbf{b}^{n+1} + \mathbf{b}^n) + \tfrac{\Delta t}{2}(\mathbf{g}^{n+1} + \mathbf{g}^n) \end{aligned} \tag{7.26}$$

As an implicit method, it allows us to take time steps whose size is of the order of the temporal variation of the magnetic driving field, which is tremendously larger than the CFL stability condition imposed on an explicit method. Moreover, the trapezoid rule has the very desirable property that the equation of global energy conservation is exactly discretized.

To solve this system we first eliminate $\mathbf{b}^{n+1}$ from (7.26) using (7.25), to obtain

$$\begin{aligned} \left[M_1(\epsilon) + \tfrac{\Delta t}{2} M_1(\sigma) + \left(\tfrac{\Delta t}{2}\right)^2 C^T M_2(\mu^{-1}) C\right]\mathbf{e}^{n+1} = \\ \left[M_1(\epsilon) - \tfrac{\Delta t}{2} M_1(\sigma) - \left(\tfrac{\Delta t}{2}\right)^2 C^T M_2(\mu^{-1}) C\right]\mathbf{e}^n + \\ \Delta t C^T M_2(\mu^{-1})\mathbf{b}^n + \tfrac{\Delta t}{2}(\mathbf{g}^{n+1} + \mathbf{g}^n) \end{aligned} \tag{7.27}$$

The solution $\mathbf{e}^{n+1}$ of this equation is then substituted into (7.25) to obtain $\mathbf{b}^{n+1}$.

7.2.4 Linear Solution

The coefficient matrix of (7.27) is symmetric, positive-definite, and we use the conjugate gradient method to solve the system. It is, however, extremely poorly conditioned when taking the relatively huge time steps we require. This is due to the fact that the term $C^T M_2(\mu^{-1}) C$, which dominates in the free-space region where

$\sigma = 0$, has a large nullspace. As a result, the convergence rate is very poor using symmetric Gauss-Seidel (GS) as a preconditioner. To remedy this we have adapted a relaxation scheme proposed by Hiptmair [27] for use as a preconditioner. In the following algorithm, G denotes the matrix representation of the gradient operator as a map from the finite dimensional spaces $\mathcal{W}^0$ to $\mathcal{W}^1$.

Hiptmair Preconditioning for $A\mathbf{e} = \mathbf{f}$:

1. Symmetric GS step for $A\mathbf{e} = \mathbf{f}$ (with 0 initial guess)
2. Transfer residual to nodes: $\mathbf{r}' \leftarrow G^T(\mathbf{f} - A\mathbf{e})$
3. Symmetric GS step on $A'\mathbf{e}' = \mathbf{r}'$, $A' \equiv G^T A G$
4. Correct edge based solution: $\mathbf{e} \leftarrow \mathbf{e} + G\mathbf{e}'$
5. Final symmetric GS step on $A\mathbf{e} = \mathbf{f}$ to symmetrize.

Chapter 8

Parallelism

This chapter presents the the parallel programming model in TRUCHAS.

8.1 Background on Parallel Programming

One goal of the TRUCHAScode is to run simulations with large data sets and complicated, coupled, physical processes. To make that possible, we developed the code so that it can use multiple computer processors simultaneously. Because multiple CPUs are working together, at the same time, on the same simulation, this is called “parallel programming”. An introduction to many of the topics discussed in this section can be found in [28].

8.1.1 Parallel Computer

A computer which supports paralell programming is often called a parallel computer. Many different types of parallel computers are in use today. While all parallel computers have multiple CPUs, one major distinction between them is whether all the CPUs have direct access to a common memory system or not. CPUs that share memory are “Shared Memory” parallel computers. Parallel computers where each CPU has direct access to only its own, local, memory system are “Distributed Memory” computers. That is, the memory is distributed among the CPUs.

8.1.2 Shades of Grey

Shared memory and distributed memory parallel computers represent two ends of a spectrum. Most modern parallel computers fall somewhere in between. Large Silicon Graphics, Inc. (SGI) systems are "distributed, shared memory". On those systems, CPUs have their own memory system, so memory is distributed among the processors. However, a combination of hardware and software allows any processor to directly read or write the memory of any other processor. So the memory is both distributed and shared.

Many of the parallel computers are built as a cluster of processing nodes. The processing nodes may themselves be shared memory parallel computers. For instance, we sometimes run on a cluster of dual-processor PCs. The Compaq computer system, the latest large scale supercomputer at Los Alamos, is a cluster of shared-memory nodes.

8.1.3 Programming for Distributed Memory Parallel Computers

TRUCHASis developed to run on distributed memory parallel computers. Every interesting, modern parallel computer can be efficiently used with that model. All non-local reading and writing of data is done through explicit library calls. (The library may make use of the fact that high-speed, shared memory hardware is available and deliver high performance, but TRUCHASdoes not assume that exists.)

There are some disadvantages to limiting ourselves to a distributed memory model. One significant disadvantage is that it is often easier to program for shared memory systems than for distributed memory systems. Another disadvantage is that shared memory systems admit higher performance algorithms than distributed memory systems. So, by rejecting a shared memory programming model we are making our development task harder, and potentially decreasing our performance.

We decided that these disadvantages were outweighed by the portability we get by assuming only distributed memory. We can run on cheaper systems, since shared memory hardware is expensive. So TRUCHAScan run on inexpensive clusters of PCs. Also, there has not yet emerged an ubiquitous shared memory programming model that delivers high performance on large systems across vendors. Our portability would be limited if we assumed one vendor's particular shared memory paradigm.

8.2 SPMD Programming Model

TRUCHASuses the SPMD (Single Program, Multiple Data) programming paradigm. We write our code using common, single processor programming languages (mostly FORTRAN95), and make calls to a library to move data between processors. The same executable runs on each processor (that's the "Single Program" part of SPDM). Each processor has different data (different regions of the mesh, for example), and hence

computes different results (that's the "Multiple Data" part of SPMD). Many of the algorithms for computing across the processors are "data parallel algorithms" [29].

8.2.1 MPI

Of course, the different processors have to exchange data in order to execute the desired algorithms. For instance, computing the gradient of a field for a cell requires field data from surrounding cells. The data for those surrounding cells may be on different processors. Therefore, we need some way to transfer data between processors. By choice we have limited ourselves to programming languages that do not know anything about moving data between processors. Therefore, we need to use a special-purpose library. The library we chose for Truchas is the Message Passing Interface (MPI) [30, 31, 32]. PVM is an alternative communication library.

8.2.2 Communication Library: PGSLib

MPI provides the fundamental capability to move data from one process to another. For instance, the routine `MPI_Send` sends a buffer of data from one processor to another. However, the MPI interface is at a very low level - it refers to real or integer buffers, word sizes, and processor numbers. TRUCHASuses a communication library, PGSLIB [33], which provides the abstractions appropriate for unstructed mesh algorithms. For instance, PGSLIBprovides routines to gather data from all surrounding cells. A developer using this routine does not have to concern themselves with the detail of where the surrounding cell data is stored - local on the same processor, or on other processors.

PGSLIBprovides abstractions for all the functionality that might require interprocessor communication. MPI routines are never called directly in TRUCHAS, only indirectly through PGSLIB . One nice benefit of that is that PGSLIBprovides a "serial emulator". That is, if somebody is running on a serial computer (that is, they are using only a single processor), by linking with the serial version of PGSLIBthen they can avoid using MPI altogether. That simplifies debugging, and also simplifies installation, since if you do not plan to run on multiple processors there is no need to install MPI.

8.3 Developing Code In Truchas

In this section we review the fundamentals for developing code in Truchas, with exclusive emphasis on assuring that the code will run properly in parallel.

There are two concepts, which, if kept clearly in mind, can assure that correct code is generated, in most

cases.

All code executes locally, on a single processor, except explicit calls into the PGSLIBlibrary.

All processors must execute the same calls, in the same order, into PGSLIB.

Code which involves all the processors (calls into PGSLIB) is called **global**. Code which involves only a single processor is called **local**. Hence, our parallel programming paradigm is often called **global–local programming**. The general flow of control involves mostly local code, interspersed with global calls into PGSLIB.

8.3.1 What Is Local Data, and What Is Global Data?

Since we are assuming distributed memory, all data is "owned" by one and only one processor. However, sometimes we want to perform global operations on what seems to be a single, global, data set.

Local data is data which is operated on only by a single processor. A temporary variable inside a loop is local. Global data is a union of local data sets, together with a (possibly conceptual) description of how the local data sets are ordered and joined.

For instance, scattering a scalar field, *e.g.* temperature, from cell centers to cell vertices is a global operation on global data. The cell centers and the cell vertices are global data sets because we want to consider the scatter operation on the whole mesh, regardless of how it may be distributed to different processors. We consider that each processor contains the data for a portion of the mesh. Clearly, the scatter operation cannot complete without all processors participating. The code for this example might be:

```
real, dimension(ncells):: Temperature
real, dimension(nnodes):: Node_Sum_Temperature

call SUM_SCATTER(Node_Avg_Temperature, Temperature)
```

`SUM_SCATTER` is a global subroutine, so this fragment of code is global.

The distinction between local data and global data is frequently one of context. For instance, if we want to scale cell centered temperature data by some constant, we might write:

```
real, dimension(ncells) :: Temperature
```

```
Temperature = scale_factor * Temperature
```

In this case Temperature is local data, since we are operating on each processor's portion of the mesh independently.

8.3.2 Compute Locally

Once again, because our machine model assumes distributed memory, and because we do not assume any native machine capability to access memory of other processors, all computation is local. That is, every processor can only compute on data which it owns, *i.e.* local data. This means that most of the Truchas code looks like single processor, serial code. The main distinction is that any operation which involves the whole mesh must use special routines. Most of the physics algorithms are local, however.

8.3.3 Communication is Global

Any non-local algorithm requires communication between processors, and hence must use one the PGSLIBroutines. Common global operations include gathering data from vertices to cell centers, scattering (with a combiner such as addition) data from cell centers to vertices and computing the dot product of two vectors. Input and output is also global.

8.3.4 Partitioning The Data

So far we have mentioned that operations on the whole mesh, such as computing the derivative of a field, are global operations. (Derivatives require neighbor cell information, and hence are non-local.) However, users supply Truchas with a single mesh, and so the question is how to transition from a single mesh to a distributed, partitioned, mesh.

We divide the mesh among processors using a mesh partitioner. In the general case we use a program called Chaco, from Sandia National Laboratory. That program looks at the mesh connectivity and returns a permutation array which tells us how to reorder the mesh so that we can get the number of partitions we want with as little communication between partitions as possible. It also tries to make the partitions as evenly sized as possible so that all processors will have the same amount of work to do. (Obviously, returning one partition containing the whole mesh and all other partitions containing no data at all would minimize communication between partitions. That would not help us distribute the computation between processors, though.)

In same cases, rather than using Chaco we use a simple algorithm which divides the mesh into blocks. This works well if the mesh is a rectilinear mesh that has been generated by Truchas' internal mesh generator. It does not work well for any other kind of mesh.

8.3.5 Common Pitfalls

The most common mistake that inhibits parallel operation is to treat a global operation as if it were local. For instance:

```
real, dimension(ncells):: Temperature
real, dimension(nnodes):: Node_Sum_Temperature

if (MAXVAL(Temperature) <= Temp_Min) then
   call SUM_SCATTER(Node_Sum_Temperature, Temperature)
end if
```

The error here is that each processor will have a different result for `MAXVAL(Temperature)`, so some processors may enter the if clause and others will not. Since `SUM_SCATTER` is a global operation, our symantics require that all processors must execute it. If some do and some do not, then the operation will not complete properly, and program will crash or hang, in the best case, or return incorrect results in the worst case.

One thing that makes this such a nefarious bug is that it shows up only when some processors enter the if clause and some do not. It is likely that for many data sets no processors or all processors will enter the if clause, and hence the code will seem to be correct. In addition, for a given data set, the bug may show up on some number of processors and not on other numbers of processors. (The bug will never show up on one processor.)

One way to find this sort of bug is with a debugger such as Totalview. If the program is crashing or hanging, you can examine it with Totalview and notice that some processors are executing one piece of code and other subroutines are hung inside the if clause.

One correct way to code this example is:

```
real, dimension(ncells):: Temperature
real, dimension(nnodes):: Node_Avg_Temperature

if (PGSLib_Global_MAXVAL(Temperature) <= Temp_Min) then
```

```
      call SUM_SCATTER(Node_Avg_Temperature, Temperature)
   end if
```

By making the MAXVAL global the same value is returned to all processors. Assuming that Temp_Min is the same on all processors, then either all processors or no processor will enter the if clause.

Appendix A

Discrete Operators

Discrete operators in TRUCHASare defined as those functions that discretely approximate continuous vector calculus operators. The four operators currently approximated are the gradient (∇), curl ($\nabla \times$), divergence ($\nabla \cdot$), and average ($\langle\rangle$) operators. The operators act on either scalar (ϕ) or vector ($\mathbf{v}$) input data and return as output either scalar or vector data, as summarized in the table below.

Table A.1: TRUCHASdiscrete operators.

Operator	Input	Output	Input Type	Output Type
gradient	ϕ	$\nabla\phi$	scalar	vector
curl	$\mathbf{v}$	$\nabla \times \mathbf{v}$	vector	vector
divergence	$\mathbf{v}$	$\nabla \cdot \mathbf{v}$	vector	scalar
average	ϕ	$\langle\phi\rangle$	scalar	scalar

A.1 Summary

The vector and scalar data input to the discrete operators itemized in Table A.1 can be located at either cell centroids or cell nodes. Similarly, the scalar or vector discrete operator output can be located at cell centroids, cell face centroids, or cell nodes, as summarized in Table A.2 below.

The location of data is indicated by the subscripts c, f, and n, for cell centroid, face centroid, and cell node, respectively.

Each discrete operator in Table A.2 above will be derived from an expanded data set if its location is on or

Table A.2: Discrete operator input/output data location.

Operator	Input	Output	Input Type	Output Type
face gradient	ϕ_c	$\nabla_f \phi_c$	scalar	vector
cell gradient	ϕ_c	$\nabla_c \phi_c$	scalar	vector
cell curl	$\mathbf{v}_c$	$\nabla_c \times \mathbf{v}_c$	vector	vector
cell divergence	$\mathbf{v}_c$	$\nabla_c \cdot \mathbf{v}_c$	vector	scalar
cell divergence	$\mathbf{v}_n$	$\nabla_c \cdot \mathbf{v}_n$	vector	scalar
face average	ϕ_c	$\langle \phi_c \rangle_f$	scalar	scalar
node average	ϕ_c	$\langle \phi_c \rangle_n$	scalar	scalar

near a boundary. If the discrete operator location coincides exactly with a cell face centroid on a boundary, then it is *on a boundary*. If the discrete operator location is *near a boundary*, then then at least one of the cells in the domain of dependence will have at least one face or node on a boundary. A cell in the domain of dependence is an immediate neighbor whose data effects the value of the discrete operator. For these cases, additional boundary condition (BC) data is used to determine the discrete operator, as itemized in the table below.

Table A.3: Conditions for including additional BC data in determining discrete operators.

Operator	Additional BC Data is Included if . . .	Source of BC Data: All boundary faces . . .
face gradient	face owns ≥ 1 boundary node	sharing ≥ 1 face node
cell gradient	cell owns ≥ 1 boundary node	of cell and neighbors
cell curl	cell owns ≥ 1 boundary node	of cell and neighbors
cell divergence	cell owns ≥ 1 boundary node	of cell and neighbors
face average	face owns ≥ 1 boundary node	sharing ≥ 1 face node
node average	node on a boundary	sharing boundary node

As is evident in Table A.3 above, BC data is included in the computation of discrete operators if the cell or face of concern is on or near a boundary. In this case, BC data comes from the boundary faces of the reference cell (or face) as well as those boundary faces of immediate neighbor cells. As an example, the influence of BC data on a cell and face gradient is depicted in the schematic below.

A.1.1 Algorithm Overview

Given discrete scalar data ϕ residing at cell centroids, approximations for first-order derivatives (e.g., $\nabla\phi$) on 2-D and 3-D fully-unstructured meshes must be made. The method is based on the work of Barth [34], who has devised innovative least squares algorithms for the linear and quadratic reconstruction of discrete data on unstructured meshes. Second (and higher) order accuracy has been demonstrated on highly irregular (e.g., random triangular) meshes. In this approach, Taylor series expansions ϕ_n^{TS} are formed from each reference cell i at centroid $\mathbf{x}_i$ to each immediate cell neighbor n at centroid $\mathbf{x}_n$:

$$\phi_n^{\mathrm{TS}}(\mathbf{x}_i) = \phi_i(\mathbf{x}_i) + (\mathbf{x}_n - \mathbf{x}_i) \cdot \nabla_i\phi_i(\mathbf{x}_i) + \dots \tag{A.1}$$

where *immediate neighbor* cells are defined to be those cells sharing at least one vertex with the reference cell i. The Taylor-Series expansion of ϕ in Equation A.1 above is termed a *linear reconstruction* if only the first derivative (gradient) terms are retained in the expansion. This is the assumption currently made in TRUCHAS. The sum $(\phi_n^{\mathrm{TS}} - \phi_i)^2$ over all n immediate neighbors is then minimized in the least squares (L_2 norm) sense:

$$\min \sum_n (\phi_n^{\mathrm{TS}} - \phi_i)^2 \Longrightarrow \sum_n (\phi_n^{\mathrm{TS}} - \phi_i) \frac{\partial(\phi_n^{\mathrm{TS}} - \phi_i)}{\partial \nabla_i\phi_i} = 0 \tag{A.2}$$

The above minimization yields $N = \texttt{ndim} * \texttt{nneighbors}$ equations for the unknown components of $\nabla_i\phi_i$. Here `nneighbors` is the total number of immediate neighbors, which includes the number of interior (`int_neighbors`) and boundary (`bc_neighbors`) neighbors, and `ndim` is the dimensionality of the system, i.e., the number of unknowns for $\nabla_i\phi_i$. Each neighbor n therefore yields `ndim` separate, linearly-independent equations for $\nabla_i\phi_i$. If $N <$ `ndim` the system is undetermined, if $N =$ `ndim`, the system is solvable, and if $N >$ `ndim`, the system is overdetermined. In general the system is overdetermined, hence a minimizing solution must be sought according to Equation A.2 above.

One way in which least squares solutions to Equation A.2 can be found by solving a linear system known as the normal equations []:

$$\left(A^{\mathrm{T}} W A\right) \mathbf{x} = A^{\mathrm{T}} W \mathbf{b}, \tag{A.3}$$

where A,

$$A = \begin{pmatrix} (x_k - x_i) & (y_k - y_i) & (z_k - z_i) \\ \vdots & \vdots & \vdots \\ (x_n - x_i) & (y_n - y_i) & (z_n - z_i) \end{pmatrix}, \tag{A.4}$$

is a dense `ndim` $\times N$ matrix, and W,

$$W = \begin{pmatrix} w_k & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & w_n \end{pmatrix}, \tag{A.5}$$

is a diagonal $N \times N$ matrix. The diagonal entries in W are individual weights $w_k = w_k^d * w_k^g$, expressed in general form as the product of a geometric weight w_k^g and a data-dependent weight w_k^d. The geometric weight w_k^g is $1/|\mathbf{x}_k - \mathbf{x}_i|^t$ (we take $t = 2$) and the data-dependent weight w_k^d, unity by default, is the optional `Weight` argument passed into all discrete operator procedures, hence its value is determined at execution time. The vector $\mathbf{b}$ (length N) is given by

$$\mathbf{b} = \begin{pmatrix} \phi_k - \phi_i \\ \vdots \\ \phi_n - \phi_i \end{pmatrix}, \tag{A.6}$$

and the solution vector $\mathbf{x}$ (length `ndim`) is

$$\mathbf{x} = \begin{pmatrix} \nabla_{xi}\phi_i \\ \nabla_{yi}\phi_i \\ \nabla_{zi}\phi_i \end{pmatrix}. \tag{A.7}$$

After carrying out the matrix-vector and matrix-matrix multiplications in Equation A.3, the linear system

$$A'\mathbf{x} = \mathbf{b}', \tag{A.8}$$

is obtained, where A' is a `ndim` $\times$ `ndim` matrix,

$$A' = \begin{pmatrix} \sum_n w_n \delta x_{ni} \delta x_{ni} & \sum_n w_n \delta y_{ni} \delta x_{ni} & \sum_n w_n \delta z_{ni} \delta x_{ni} \\ \sum_n w_n \delta x_{ni} \delta y_{ni} & \sum_n w_n \delta y_{ni} \delta y_{ni} & \sum_n w_n \delta z_{ni} \delta y_{ni} \\ \sum_n w_n \delta x_{ni} \delta z_{ni} & \sum_n w_n \delta y_{ni} \delta z_{ni} & \sum_n w_n \delta z_{ni} \delta z_{ni} \end{pmatrix}, \tag{A.9}$$

where $\delta x_{ni} = x_n - x_i$, $\delta y_{ni} = y_n - y_i$, and $\delta z_{ni} = z_n - z_i$. The vector $\mathbf{b}'$ (length `ndim`) is given by

$$\mathbf{b}' = \begin{pmatrix} \sum_n w_n \delta x_{ni} \delta \phi_{ni} \\ \sum_n w_n \delta y_{ni} \delta \phi_{ni} \\ \sum_n w_n \delta z_{ni} \delta \phi_{ni} \end{pmatrix}, \tag{A.10}$$

where $\delta\phi_{ni} = \phi_n - \phi_i$. Each component of the A' and $\mathbf{b}'$ is therefore derived by summing over all immediate interior and boundary neighbors (`nneighbors`) of the reference cell i. The resulting `ndim` $\times$ `ndim` linear system is easily solved with conventional direct methods such as LU decomposition.

If the reference value of ϕ, e.g., ϕ_i in Equation A.1, is not known, then Equation A.2 is derived by performing partial derivatives with respect to ϕ_i in addition to the components of $\nabla\phi$. This results in `ndim`$+1$ unknowns, hence ϕ_i will also be part of the solution. This is the case for face gradients, where ϕ_i is ϕ_f, the unknown value of ϕ at face f. Face values of ϕ are not known because discrete values of ϕ reside only at cell centroids.

$$A' = \begin{pmatrix} \sum_n w_n \delta x_{ni} \delta x_{ni} & \sum_n w_n \delta y_{ni} \delta x_{ni} & \sum_n w_n \delta z_{ni} \delta x_{ni} & \sum_n w_n \delta x_{ni} \\ \sum_n w_n \delta x_{ni} \delta y_{ni} & \sum_n w_n \delta y_{ni} \delta y_{ni} & \sum_n w_n \delta z_{ni} \delta y_{ni} & \sum_n w_n \delta y_{ni} \\ \sum_n w_n \delta x_{ni} \delta z_{ni} & \sum_n w_n \delta y_{ni} \delta z_{ni} & \sum_n w_n \delta z_{ni} \delta z_{ni} & \sum_n w_n \delta z_{ni} \\ \sum_n w_n \delta x_{ni} & \sum_n w_n \delta y_{ni} & \sum_n w_n \delta z_{ni} & \sum_n w_n \end{pmatrix}, \tag{A.11}$$

$$\mathbf{b}' = \begin{pmatrix} \sum_n w_n \delta x_{ni} \phi_n \\ \sum_n w_n \delta y_{ni} \phi_n \\ \sum_n w_n \delta z_{ni} \phi_n \\ \sum_n w_n \phi_n \end{pmatrix}, \tag{A.12}$$

$$\mathbf{x} = \begin{pmatrix} \nabla_{xi} \phi_i \\ \nabla_{yi} \phi_i \\ \nabla_{zi} \phi_i \\ \phi_i \end{pmatrix}. \tag{A.13}$$

Least squares reconstruction methods are quite powerful and attractive for a number of reasons. First, they are not married to any particular mesh topology or dimensionality, hence are easily amenable to any unstructured mesh in one, two, or three dimensions. All that is required is a set of discrete data points described by their data values and their physical location. Second, there are no constraints (other than conservation of the mean, as described by Gooch [35]) on what has to be minimized or how that minimization is to be performed. For example, L_1 or L_∞ norms might also be minimized rather than the L_2 norm as above. Third, the discrete data points can be arbitrarily weighted and/or constrained in the minimization process, which is apparent in Equation A.3 via inclusion of the geometric weights w_k. Data-dependent weights can also be included. These might arise as a result of constraints such as monotonicity, validity, etc. With data-dependent weight, however, the resulting overdetermined system of equations frequently has to be solved with a method other than the normal equations. As a final note, the accuracy of this method is easily increased by including additional terms in the Taylor series expansion ϕ_i^{TS}.

Appendix B

Support-Operators

The support-operators method [36, 37] is a means of constructing discrete analogs of invariant differential operators, like the divergence and the gradient. In TRUCHAS, a mixed hybrid formulation of the support-operators method is used within the species diffusion component. It is also used as a discretization scheme for the electromagnetics component (disguised as finite elements). The heat transfer component and the projection step in the fluid flow component, uses the support-operators discretization of the AUGUSTUS code package [38, 39, 37].

There are several formulations of the support-operators methodology, and the version used by the species diffusion and electromagnetics component are different from the version used by the heat transfer and and the fluid flow components. We include both formulations here for elucidation and comparison.

B.1 Species Diffusion Component Support-Operators Formulation

This formulation is used for the species diffusion component and its description was adapted from [40].

To provide an overview of support-operators, we consider the following standard diffusion equation given by

$$-\mathrm{div}\,(K\,\mathbf{grad}\,T) = f \;\; \forall\,(x,y,z) \in \Omega \subset \Re^3, \tag{B.1}$$

where $T(x,y,z)$ represents concentration, $K(x,y,z)$ is the conductivity, and f is a source term. In general, K may be a symmetric positive-definite tensor that can vary discontinuously in space. The problem becomes well-posed when we enforce boundary conditions. Here we only consider homogeneous Dirichlet boundary conditions, i.e.,

$$T(x,y,z) = 0, \;\; \forall (x,y,z) \in \Omega. \tag{B.2}$$

Next, we reduce the second-order operator given in (B.1) to a more natural set of first-order operators, given by

$$\operatorname{div} \mathbf{W} = f, \qquad \mathbf{W} = -K \, \mathbf{grad}\, T, \tag{B.3}$$

where the first equation is an expression of conservation and the second is an expression of *Fourier's Law*.

In the **first step** of the support-operators discretization method, we specify discrete degrees of freedom for the scalar and vector unknowns. With TRUCHAS in mind, we assume a mesh consisting of hexahedral elements. Let Ω^h be such a mesh consisting of hexahedral elements $\{H_c\}_{c=1}^{ncells}$ such that

$$\Omega = \bigcup_{c=1,ncells} H_c.$$

We use the geometric cell- and face-centers to define the locations of the scalar and vector unknowns, respectively. Specifically, the face-centered vector unknowns are represented only by their normal component $\mathbf{W} \cdot \mathbf{n}$.

In the **second step** of the support operator discretization method, we equip the discrete spaces HC and $\mathcal{HF}$ of scalar and vector unknowns, respectively, with scalar products. These scalar products are simple extensions of their continuous analogues. We define the the inner product for discrete scalar functions u and v in HC as

$$[u, v]_{HC} \stackrel{\text{def}}{=} \sum_{c=1}^{ncells} u_c \, v_c \, |H_c|, \tag{B.4}$$

where u_c and v_c are the cell-centered values, and $|H_c|$ is the element volume. For the space of discrete vector unknowns, the scalar product is not as straightforward and we provide only a brief description here. See [36] for a more careful description of the vector unknown space scalar product. Nonetheless, we define the inner product by

$$[\mathbf{A}, \mathbf{B}]_{\mathcal{HF}} \stackrel{\text{def}}{=} \sum_{H_c \in \Omega^h} [\mathbf{A}, \mathbf{B}]_{\mathcal{HF}, H_c} \tag{B.5}$$

with

$$[\mathbf{A}, \mathbf{B}]_{\mathcal{HF}, H_c} \stackrel{\text{def}}{=} \sum_{n=1}^{8} (K_n^c)^{-1} (\mathbf{A}_n^c, \mathbf{B}_n^c) \, V_n^c. \tag{B.6}$$

Here, n loops over all vertices of the hexahedron H_c, and K_n^c is the diffusion coefficient local to H_c at its vertex n. V_n^c denotes a "nodal" volume, and $\mathbf{A}_n^c$ and $\mathbf{B}_n^c$ denote cartesian vectors located at vertex n of H_c. These nodal vectors are constructed using face-centered vector unknowns from faces that are adjacent to the node and belong to H_c.

The **third step** of the support operator discretization method is to define a discrete analogue of the divergence operator $\mathbf{DIV} : \mathcal{HF} \rightarrow \mathcal{HC}$. Based on Gauss' divergence theorem, we have the coordinate invariant definition of div given by

$$\mathrm{div}\,\mathbf{W} = \lim_{V \to 0} \frac{1}{V} \oint_{\partial V} \mathbf{W} \cdot \mathbf{n}\, \mathrm{d}S. \tag{B.7}$$

We use this expression to define a discrete divergence over a hexahedron in Ω^h. Denote by F a particular face with area A_F. Then the discrete analogue of (B.7) on hexahedron H_c is

$$(\mathbf{DIV}\ \mathbf{W})_{H_c} = \frac{1}{|H_c|} \sum_{F \in H_c} A_F\, \mathbf{W} \cdot \mathbf{n}_{F,c}, \tag{B.8}$$

where $\mathbf{n}_{F,c}$ is the outward normal to face F of hexahedron H_c.

Finally, in the **fourth step** of the support operator discretization method, we derive a discrete flux operator $\mathcal{G}$ (as the discrete analogue of the operator $-K\,\nabla$) that is adjoint to the discrete divergence operator $\mathbf{DIV}$ with respect to the two scalar products (B.4) and (B.5), i.e.

$$[\mathbf{DIV}W, u]_{\mathcal{HF}} = [W, \mathcal{G}u]_{HC}\,, \quad \forall W \in \mathcal{HF},\ \forall u \in HC. \tag{B.9}$$

To derive an explicit formula for $\mathcal{G}$, we consider an auxiliary scalar product $< \cdot, \cdot >$ and relate it to scalar products (B.4) and (B.5). Denote by $< \cdot, \cdot >$ the standard vector dot product, then

$$[u, v]_{HC} =< \mathcal{D}u, v >, \quad \text{and } [U, W]_{\mathcal{HF}} =< \mathcal{M}U, W >,$$

where $\mathcal{D}$ is a diagonal matrix $\mathcal{D} = \mathrm{diag}\{|H_1|, |H_2|, \ldots, |H_{ncells}|\}$, with the volumes of all hexahedrons on the diagonal, and $\mathcal{M}$ is a sparse symmetric mass matrix. Combining these last two formulas, we get

$$[W, \mathbf{DIV}^\star u]_{\mathcal{HF}} =< W, \mathcal{M}\mathbf{DIV}^\star u >= [\mathbf{DIV}W, u]_{HC} =< W, \mathbf{DIV}^T \mathcal{D}u > .$$

for all $W \in \mathcal{HF}$ and $u \in HC$. Here, $\mathbf{DIV}^T$ is the adjoint of $\mathbf{DIV}$ with respect to the auxiliary scalar product. Therefore,

$$\mathcal{M}\mathbf{DIV}^\star = \mathbf{DIV}^T \mathcal{D}$$

which implies

$$\mathcal{G} = \mathbf{DIV}^\star = \mathcal{M}^{-1}\mathbf{DIV}^T \mathcal{D}. \tag{B.10}$$

The support operator discretization of the first order system (B.3) is now given by

$$\mathbf{DIV}W = f^h, \quad \text{and } W = \mathcal{G}T,$$

where $W \in \mathcal{HF}$, $T \in HC$, and f^h is a vector of integral averages of f, each component taken over one hexahedron in the mesh.

Note that $\mathbf{DIV}^T u$ is defined on faces and represents the difference in concentration between two adjacent cells. This operator forms the basis of the gradient in the ORTHO algorithm used in heat transfer calculations.

B.1.1 Mixed Hybrid Formulation

In the mixed hybrid formulation, we localize the face centered vector unknowns from step two above to cells by introducung two separate ones, one for each cell adjacent to a face. Continuity of the normal flux across a face is enforced by introducing additional equality constraints: We require the face flux on a face belonging to a cell to equal that of the equivalent face on its neighbor cell. These additional equality constraints are introduced by means of Lagrange multipliers which results in additional scalar degrees of freedom, one for each face. It turns out that these unknowns represent in fact the concentration on faces.

The algebraic equations that must be solved can be written as a 3×3 block matrix, where the blocks derive from dividing the unknowns into cell centered concentrations, face centered concentrations, and face centered normal fluxes. The mixed hybrid linear system of equations arises when the face centered normal flux variables are eliminated (via block elimination). We go one step further and then eliminate the face centered concentration variables as well.

B.2 AUGUSTUS Support-Operators Formulation

This formulation is used for the heat transfer component and the fluid flow component of Truchas. The following description was adapted from [38, 39].

We start with the following diffusion equation

$$-\text{div}\, D\mathbf{grad}\,\phi = S\,, \tag{B.11}$$

which can be written

$$\text{div}\ \overrightarrow{F} = S\,, \tag{B.12}$$

$$\overrightarrow{F} = -D\mathbf{grad}\,\phi\,, \tag{B.13}$$

where ϕ represents the temperature, D represents the thermal conductivity, $\overrightarrow{F}$ represents the heat flux and S represents a heat source in the heat transfer solution. In the projection step of the fluid flow solution, ϕ represents a change in the pressure, D represents the inverse of the density, $\overrightarrow{F}$ represents an "acceleration" flux and S represents an "acceleration gradient" source.

To discretize this equation, we integrate over a cell (c), applying Gauss's Theorem to change the volume

integral into a surface integral over the faces (f):

$$\sum_f \overrightarrow{F_{c,f}} \cdot \overrightarrow{A_{c,f}} = S_c V_c \tag{B.14}$$

This is the basic form used by TRUCHAS. The gradient terms ($\overrightarrow{F_{c,f}} \cdot \overrightarrow{A_{c,f}}$) are calculated separately from the solution of the conservation equation, so that a matrix-free solution procedure may be followed. This matrix-free solution method requires that the gradient terms be calculated when given the ϕ-values of a trial solution, so that is the form derived by the Support Operator Method.

Note that this discretization will be inherently conservative, and that no derivatives are taken across material boundaries – a rigorous treatment. We locate unknowns for ϕ at the cell centers and the cell faces. The $\overrightarrow{F_{c,f}} \cdot \overrightarrow{A_{c,f}}$ (gradient) terms, on each face of a cell, must be defined in terms of the ϕ's within that cell.

In summary, the Support Operator Method represents the diffusion term ($\mathrm{div}\, D\mathbf{grad}\,\phi$) as the divergence ($\mathrm{div}$) of a gradient ($\mathbf{grad}$), explicitly defines one of the operators (in this case, the divergence operator), and then defines the remaining operator (in this case, the gradient operator) as the discrete adjoint of the first operator. The last step is accomplished by discretizing a portion of a vector identity. In other words, the first operator is set up explicitly, and the second operator is defined in terms of the first operator's definition. The rest of this section derives the Support Operator Method in more detail.

To derive the Support Operator Method used in AUGUSTUS, we start with this vector identity,

$$\mathrm{div}\left(\phi\overrightarrow{W}\right) = \phi \mathrm{div}\,\overrightarrow{W} + \overrightarrow{W}\cdot\mathbf{grad}\,\phi\,, \tag{B.15}$$

where ϕ is the scalar variable to be diffused and $\overrightarrow{W}$ is an arbitrary vector, and integrate over a cell volume:

$$\int_c \mathrm{div}\left(\phi\overrightarrow{W}\right)\,dV = \int_c \phi \mathrm{div}\,\overrightarrow{W}\,dV + \int_c \overrightarrow{W}\cdot\mathbf{grad}\,\phi\,dV\,. \tag{B.16}$$

Each term in the equation above will be treated separately.

The first term in Equation B.16 can be transformed via Gauss's Theorem into a surface integral,

$$\int_c \mathrm{div}\left(\phi\overrightarrow{W}\right)\,dV = \oint_S \left(\phi\overrightarrow{W}\right)\cdot\overrightarrow{dA}\,. \tag{B.17}$$

This is discretized into values defined on each face of the hexahedral cell,

$$\oint_S \left(\phi\overrightarrow{W}\right)\cdot\overrightarrow{dA} \approx \sum_f \phi_f \overrightarrow{W_f}\cdot\overrightarrow{A_f}\,. \tag{B.18}$$

The second term in Equation B.16 is approximated by first assuming that ϕ is constant over the cell (at the center value), and then performing a discretization similar to the previous one for the first term:

$$\int_c \phi \mathrm{div}\, \overrightarrow{W}\, dV \approx \phi_c \int_c \mathrm{div}\, \overrightarrow{W}\, dV \,, \tag{B.19}$$

$$= \phi_c \oint_S \overrightarrow{W} \cdot \overrightarrow{dA} \,, \tag{B.20}$$

$$\approx \phi_c \sum_f \overrightarrow{W_f} \cdot \overrightarrow{A_f} \,. \tag{B.21}$$

Turning to the final (third) term in Equation B.16, we insert the definition of the flux,

$$\overrightarrow{F} = -D\mathbf{grad}\,\phi \,, \tag{B.22}$$

to get

$$\int_c \overrightarrow{W} \cdot \mathbf{grad}\,\phi\, dV = -\int_c D^{-1}\overrightarrow{W} \cdot \overrightarrow{F}\, dV \,. \tag{B.23}$$

Note that by defining the flux in terms of the remainder of the equation, the gradient is being defined in terms of the divergence.

The third term is discretized by evaluating the integrand at each of the cell nodes (octants of the hexahedral cells, represented by n) and summing:

$$-\int_c D^{-1}\overrightarrow{W} \cdot \overrightarrow{F}\, dV \approx -\sum_n D_n^{-1}\overrightarrow{W_n} \cdot \overrightarrow{F_n} V_n \,. \tag{B.24}$$

Combining all of the discretized terms of Equation B.16 and changing to a linear algebra representation gives

$$\sum_f \phi_f \mathbf{W}_f^{\mathrm{T}} \mathbf{A}_f = \phi_c \sum_f \mathbf{W}_f^{\mathrm{T}} \mathbf{A}_f - \sum_n D_n^{-1} \mathbf{W}_n^{\mathrm{T}} \mathbf{F}_n V_n \,. \tag{B.25}$$

Rearranging terms gives

$$\sum_n D_n^{-1} \mathbf{W}_n^{\mathrm{T}} \mathbf{F}_n V_n = \sum_f (\phi_c - \phi_f)\, \mathbf{W}_f^{\mathrm{T}} \mathbf{A}_f \,. \tag{B.26}$$

Note that the right hand side is a sum over the six faces, but the left hand side is a sum over the eight nodes.

In order to express the node-centered vectors, $\mathbf{W}_n$ and $\mathbf{F}_n$, in terms of their face-centered counterparts, define

$$\mathbf{J}_n^{\mathrm{T}}\mathbf{W}_n \equiv \begin{bmatrix} \mathbf{W}_{f1}^{\mathrm{T}}\mathbf{A}_{f1} \\ \mathbf{W}_{f2}^{\mathrm{T}}\mathbf{A}_{f2} \\ \mathbf{W}_{f3}^{\mathrm{T}}\mathbf{A}_{f3} \end{bmatrix} , \tag{B.27}$$

where $f1$, $f2$, and $f3$ are the faces adjacent to node n and the Jacobian matrix is the square matrix given by

$$\mathbf{J}_n = \begin{bmatrix} \mathbf{A}_{f1} & \mathbf{A}_{f2} & \mathbf{A}_{f3} \end{bmatrix} . \tag{B.28}$$

Using this definition for the node-centered vectors $\mathbf{W}_n$ and $\mathbf{F}_n$ and performing some algebraic manipulations results in

$$\sum_n D_n^{-1} V_n \begin{bmatrix} \mathbf{W}_{f1}^{\mathrm{T}}\mathbf{A}_{f1} \\ \mathbf{W}_{f2}^{\mathrm{T}}\mathbf{A}_{f2} \\ \mathbf{W}_{f3}^{\mathrm{T}}\mathbf{A}_{f3} \end{bmatrix}^{\mathrm{T}} \mathbf{J}_n^{-1}\mathbf{J}_n^{-\mathrm{T}} \begin{bmatrix} \mathbf{F}_{f1}^{\mathrm{T}}\mathbf{A}_{f1} \\ \mathbf{F}_{f2}^{\mathrm{T}}\mathbf{A}_{f2} \\ \mathbf{F}_{f3}^{\mathrm{T}}\mathbf{A}_{f3} \end{bmatrix} = \widetilde{\mathbf{W}}^{\mathrm{T}}\widetilde{\mathbf{\Phi}} , \tag{B.29}$$

where the sum over faces has been written as a dot product of $\widetilde{\mathbf{W}}$ and $\widetilde{\mathbf{\Phi}}$, which are defined by

$$\widetilde{\mathbf{W}} = \begin{bmatrix} \mathbf{W}_1^{\mathrm{T}}\mathbf{A}_1 \\ \mathbf{W}_2^{\mathrm{T}}\mathbf{A}_2 \\ \mathbf{W}_3^{\mathrm{T}}\mathbf{A}_3 \\ \mathbf{W}_4^{\mathrm{T}}\mathbf{A}_4 \\ \mathbf{W}_5^{\mathrm{T}}\mathbf{A}_5 \\ \mathbf{W}_6^{\mathrm{T}}\mathbf{A}_6 \end{bmatrix} , \quad \widetilde{\mathbf{\Phi}} = \begin{bmatrix} (\phi_c - \phi_1) \\ (\phi_c - \phi_2) \\ (\phi_c - \phi_3) \\ (\phi_c - \phi_4) \\ (\phi_c - \phi_5) \\ (\phi_c - \phi_6) \end{bmatrix} . \tag{B.30}$$

To convert the short vectors involving the faces adjacent to a particular node into sparse long vectors involv-

ing all of the faces of the cell, define permutation matrices for each node, $\mathbf{P}_n$, such that

$$\begin{bmatrix} \mathbf{W}_{f1}^{\mathrm{T}}\mathbf{A}_{f1} \\ \mathbf{W}_{f2}^{\mathrm{T}}\mathbf{A}_{f2} \\ \mathbf{W}_{f3}^{\mathrm{T}}\mathbf{A}_{f3} \end{bmatrix} = \mathbf{P}_n \begin{bmatrix} \mathbf{W}_1^{\mathrm{T}}\mathbf{A}_1 \\ \mathbf{W}_2^{\mathrm{T}}\mathbf{A}_2 \\ \mathbf{W}_3^{\mathrm{T}}\mathbf{A}_3 \\ \mathbf{W}_4^{\mathrm{T}}\mathbf{A}_4 \\ \mathbf{W}_5^{\mathrm{T}}\mathbf{A}_5 \\ \mathbf{W}_6^{\mathrm{T}}\mathbf{A}_6 \end{bmatrix} = \mathbf{P}_n\widetilde{\mathbf{W}} \ , \tag{B.31}$$

where, for example,

$$\mathbf{P}_n = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad \begin{array}{l} \text{if } f1\,(n) = 3, \\ f2\,(n) = 5, \\ \text{and } f3\,(n) = 2. \end{array} \tag{B.32}$$

Note that $\mathbf{P}_n$ is rectangular, with a size of $N_{\textit{dimensions}} \times N_{\textit{local faces}}$ (3×6 for 3-D, 2×4 for 2-D, 1×2 for 1-D).

Using the permutation matrices, and defining $\widetilde{\mathbf{F}}$ in a fashion similar to $\widetilde{\mathbf{W}}$ ($\widetilde{\mathbf{F}}$ is a vector of $\mathbf{F}_f^{\mathrm{T}}\mathbf{A}_f$ for each cell face), gives

$$\sum_n D_n^{-1}V_n\widetilde{\mathbf{W}}^{\mathrm{T}}\mathbf{P}_n^{\mathrm{T}}\mathbf{J}_n^{-1}\mathbf{J}_n^{-\mathrm{T}}\mathbf{P}_n\widetilde{\mathbf{F}} = \widetilde{\mathbf{W}}^{\mathrm{T}}\widetilde{\mathbf{\Phi}} \ , \tag{B.33}$$

or

$$\widetilde{\mathbf{W}}^{\mathrm{T}}\left[\sum_n D_n^{-1}V_n\mathbf{P}_n^{\mathrm{T}}\mathbf{J}_n^{-1}\mathbf{J}_n^{-\mathrm{T}}\mathbf{P}_n\right]\widetilde{\mathbf{F}} = \widetilde{\mathbf{W}}^{\mathrm{T}}\widetilde{\mathbf{\Phi}} \ , \tag{B.34}$$

or

$$\widetilde{\mathbf{W}}^{\mathrm{T}}\mathbf{S}\widetilde{\mathbf{F}} = \widetilde{\mathbf{W}}^{\mathrm{T}}\widetilde{\mathbf{\Phi}} \ , \tag{B.35}$$

where

$$\mathbf{S} = \sum_n D_n^{-1}V_n\mathbf{P}_n^{\mathrm{T}}\mathbf{J}_n^{-1}\mathbf{J}_n^{-\mathrm{T}}\mathbf{P}_n \ . \tag{B.36}$$

The original vector $\overrightarrow{W}$ (on which $\mathbf{W}_f$ and $\widetilde{\mathbf{W}}$ are based) was an arbitrary vector. It can now be eliminated from the equation to give

$$\mathbf{S}\widetilde{\mathbf{F}} = \widetilde{\mathbf{\Phi}} \ . \tag{B.37}$$

This equation could be easily inverted ($\mathbf{S}$ is 6×6 in 3-D), and the resultant relationships between $\overrightarrow{F_{c,f}} \cdot \overrightarrow{A_{c,f}}$ and ϕ for each cell combined using Equation B.14 and flux equality at each face into a sparse matrix for the entire problem, which could be solved once for the solution to the original Equation B.11. This is what is done inside AUGUSTUS.

However, due to historical reasons, TRUCHAS takes a different approach. TRUCHAS combines all of the cell equations like Equation B.37 into a single (block-diagonal) equation. When the outer nonlinear iteration needs the gradients ($\overrightarrow{F_{c,f}} \cdot \overrightarrow{A_{c,f}}$), this block-diagonal matrix equation is solved. Note that no conservation equation is solved when determining these gradients, and the fluxes on either side of a face are not set equal. Also, instead of iteratively solving for the unknown Φ values on the faces, these are set to the cell-center value across the face (i.e. the neighbor cell value) once at the beginning of the inner matrix solve for the gradients. The TRUCHAS solution method will be changed in the future.

Appendix C

Linear Solution Methods

An incompressible flow algorithm constructed with the fractional-step projection method discussed in Chapter 3 requires solutions to linear systems of equations. A solution to a pressure projection equation, Equation 3.62 where P is the pressure, are required. The equation is basically elliptic in nature, and can be expressed in matrix notation as

$$\mathbf{A}\mathbf{x} = \mathbf{b}\,, \tag{C.1}$$

where $\mathbf{A}$ is a matrix resulting from the discretization, $\mathbf{x}$ is the solution vector, and $\mathbf{b}$ is a vector source term. Since for our equations the matrix $\mathbf{A}$ arises from finite volume discretizations of the Laplacian, we expect $\mathbf{A}$ to be sparse, positive definite ($\mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x} > 0$), and in general symmetric, hence our solution methods should take advantage of this structure. The total computational effort of our fractional-step scheme will be dominated by the effort required to find solutions to Equation C.1, therefore designing an efficient and scalable method for solving these systems of linear equations is of paramount importance.

Which method for the solution of Equation C.1 is recommended? There are several metrics one must take into account when considering a solution method for linear systems of equations: robustness, or ability to converge; efficiency, or convergence rate (if the method is iterative); scalability of the computational effort (relative to the number of unknowns N) required to find a solution; and complexity of implementation. Ideally, we desire a method that *always* converges (provided our equations are well-posed), that exhibits computational effort scaling linearly with N, and that requires *grid-independent iterations to convergence*. This last requirement may be restated as requiring our method to converge to a solution for a given physical domain in an iteration count that does not change with the number of grid points used to partition the domain. Of the possible solution methods briefly mentioned here, including direct and stationary iterative methods, Krylov subspace methods, multigrid methods, and hybrid methods, only the multigrid and hybrid methods have shown promise in meeting all of our requirements.

C.1 Direct and Stationary Iterative Methods

Since $\mathbf{A}$ for our equations is not dense, but rather sparse and usually diagonally-dominant, direct solution methods such as Gaussian elimination and Cholesky or LU factorization are not recommended. Because of the non-constant coefficients of the operators, FFT or cyclic reduction methods are also not effective. These methods require computational effort that scale like N^3, which can improve to N^2 if the solution method takes advantage of the sparsity of $\mathbf{A}$, but this scaling is not the linear scaling we desire. The primary attractiveness of direct solvers is their robustness, or ability to find a solution for any nonsingular $\mathbf{A}$, e.g., especially when $\mathbf{A}$ has a high condition number (ratio of maximum to minimum eigenvalues) arising from small mesh spacings or high density ratio flows. Stationary iterative methods, such as Jacobi, Gauss-Seidel, and symmetric successive over-relaxation (SSOR) [41], are attractive because of their ease of implementation, but they exhibit poor scaling, requiring computational effort that scales like N^2, and they can frequently exhibit a lack of robustness (inability or slowness to converge). While these stationary iterative methods are not recommended for finding solutions to Equation C.1, they do remain quite useful as preconditioners in Krylov subspace methods or as smoothers in the multigrid method. Further information about direct and stationary iterative methods can be found in a host of classic textbooks. I have found references [42, 43, 44] to be particularly useful.

C.2 Krylov Subspace Methods

Krylov subspace methods [44, 45] are iterative methods in which solutions to Equation C.1 are extracted from a subspace by imposing constraints on the residual vector $\mathbf{b} - \mathbf{A}\mathbf{x}$, typically that it be orthogonal to m linearly independent vectors in the subspace. The most popular and widely used Krylov subspace methods are the conjugate gradient (CG) algorithm if our positive definite matrix $\mathbf{A}$ is symmetric or generalized minimal residual (GMRES) algorithm if $\mathbf{A}$ is not symmetric. These methods are in general robust, with the CG method theoretically guaranteed to converge in N iterations, and GMRES usually able to converge if $\mathbf{A}$ does not have an excessive condition number and is diagonally-dominant. Krylov subspace methods are also relatively easy to implement.

The problem, however, is that Krylov subspace methods can be slow to converge unless the linear system given by Equation C.1 is first "preconditioned" with a preconditioning matrix $\mathbf{M}$ that either multiplies the system from the left side (left preconditioning),

$$(\mathbf{M}^{-1}\mathbf{A})\mathbf{x} = \mathbf{M}^{-1}\mathbf{b}\,, \tag{C.2}$$

or from the right side (right preconditioning),

$$(\mathbf{A}\mathbf{M}^{-1})\mathbf{y} = \mathbf{b}; \quad \mathbf{y} = \mathbf{M}\mathbf{x}\,. \tag{C.3}$$

The net effect of preconditioning is a linear operator (in parentheses above) that is closer to the identity matrix, which accelerates the convergence of the Krylov subspace method. The new linear operator, $\mathbf{A}\mathbf{M}^{-1}$

or $\mathbf{M}^{-1}\mathbf{A}$, will have a smaller condition number and eigenvalues that are more clustered than with $\mathbf{A}$ alone. Choosing the preconditioning matrix $\mathbf{M}$ is often *not* easy, as it must be an approximation to $\mathbf{A}$ that is easily inverted. Unfortunately, preconditioning the Krylov subspace method is almost always necessary, as without it these methods converge too slowly. With preconditioning, one must also solve an additional preconditioning equation, given by $\mathbf{Mz} = \mathbf{r}$, where $\mathbf{z}$ is a Krylov vector and $\mathbf{r}$ is a residual. Fortunately, approximate solutions for $\mathbf{z}$ are usually good enough, hence using simple iterative methods like SSOR or Jacobi to find $\mathbf{z}$ is often adequate.

If a good preconditioning matrix $\mathbf{M}$ can be chosen, preconditioned Krylov subspace methods can be quite powerful and efficient linear solution methods. They have been perhaps the most popular choice for the past two decades, primarily because of their robustness and ease of implementation. The problem, however, is that preconditioned Krylov subspace methods do not exhibit the scaling we desire, requiring computational work that scales like $N^{5/4}$ (at best) and iterations to convergence somewhere between $N^{1/4}$ and $N^{1/2}$. Can this scaling problem of Krylov subspace methods be overcome? Perhaps, if one is willing to focus efforts on the preconditioner, as discussed in Section C.4 below.

Additional information on Krylov space methods can be found in [46], where detailed algorithm templates sufficient for implementation are provided. See also reference [47] for an insightful introductory overview and [48, 49] for performance comparisons on linear systems arising from the NS equations.

C.3 Multigrid Methods

As stated previously, we seek a linear solution algorithm that requires computational work scaling like N, and we furthermore require that the method can find solutions in an iteration count that does not change with N. These requirements are usually met for linear elliptic solutions with a multigrid (MG) method [50], hence the scalability of the MG method is a powerful attraction. The basic premise of the MG method is the identification and suppression of long wavelength (low frequency) error modes in the residual via solutions of an equivalent linear system on a series of grids coarser than the base (finest) grid. This is in contrast to traditional iterative (Jacobi, SSOR, Gauss-Seidel) and Krylov subspace methods, which quickly eliminate only those high frequency error modes indicative of coupled nearest neighbor cells. Low frequency error modes, however, tend to persist without elimination until enough iterations have taken place for the long wavelength modes to be "seen". MG methods, on the other hand, immediately "see" these long wavelength modes on the coarser grids, which, once identified, can be suppressed after transfer back to the series of finer grids.

On each grid in the MG method, iterative approximate solutions to the linear system are obtained; rigorous solutions are not obtained on any one grid, but rather obtained on the base (finest) grid after many fine-to-coarse-to-fine (V) cycles. Space does not permit a detailed discussion of this powerful technique, but overviews can be found in the excellent monograph of Briggs [51] and the introductory textbook by Wesseling [52]. The reader is also encouraged to consult references [53, 18] for examples of the MG method

applied to linear systems of equations arising specifically from incompressible interfacial flows.

One of the most important and difficult tasks in formulating an MG algorithm is the approximation of the intergrid transfer functions, namely the restriction (fine-to-coarse) and prolongation (coarse-to-fine) operators. Performance of the MG method, measured as scalability and convergence robustness, depends crucially upon the choice of these operators. One approach is using the intergrid transfer functions to define variational or Galerkin coarse grid operators. This task can be complicated and expensive, however, especially if the restriction and prolongation operators are anything but piecewise constant (e.g., stencil growth can occur). This fact has motivated others [54, 55] to adopt the simpler approaches like those suggested in [56].

Experience has shown that the MG method at times lacks robustness, having a propensity to fail and/or exhibit slow convergence on the types of linear systems arising in incompressible interfacial flows. Here "tough" systems are those resulting from flows possessing large, abrupt, and localized changes in density and/or surface tension along an interface that is topologically complex. This lack of robustness can usually be traced to restriction and prolongation operators that are misrepresenting important interfacial physics because of inaccurate or inappropriate interpolation and/or smoothing functions. Robustness and convergence can be enhanced in many cases with more intelligent restriction and prolongation operators that do not incorrectly smooth across interfaces. Formulation and implementation of such operators in the presence of arbitrarily complex interface topologies, however, can be very expensive and cumbersome.

C.4 Hybrid Methods

Until the MG method can be made more robust and easily implemented, and Krylov subspace methods more scalable, more and more researchers are devising unified, *hybrid* methods aimed at combining the strengths of both methods while eliminating their weaknesses. Two basic types of hybrid methods have appeared in the literature to date. In the first approach, MG is the principal solution method, but a Krylov subspace method is used for solutions on one or more of the (coarser) grids rather than a simple iterative method. This approach helps to alleviate the MG robustness problem by relying on a robust Krylov method. In the second approach, a preconditioned Krylov subspace method is retained as the principal solution algorithm, but an MG-like (*multilevel*) method is used to obtain solutions to the preconditioning equation. Both approaches have merit and have exhibited improved performance, but it is not clear at this time which hybrid method exhibits the desired scalable performance without loss of robustness. One issue has become clear, however: scalable performance absolutely requires a multilevel algorithm, i.e., ideas inherent in the MG method *must* pervade.

The idea of using a symmetric MG algorithm to precondition a standard Krylov subspace (CG) method was first proposed and demonstrated by Kettler [57]. This idea, however, did not gain acceptance and popularity until the recent work of Tatebe [58]. Its use has since exploded, having shown utility in modeling incompressible flows [55, 59], semiconductor performance [60], and groundwater flow [61]. It possesses the robustness lacking in many MG algorithms, able to find solutions on the most challenging of interfacial flow

problems. A hybrid "MGCG" method, while usually scaling like a MG algorithm, occasionally exhibits CG-like scaling, hence additional research is needed to understand which aspects of the algorithm hinder consistent MG-like scaling. For most of the flow problems tested, the hybrid MGCG method requires less computational work than the MG method to seek a solution. This same basic approach was recently shown to be effective for the parallel solution of linear systems of equations on 3-D unstructured meshes [62]. In this case, a combined additive/multiplicative Schwarz [63, 45] technique was ideal for providing a means by which multilevel solutions to the preconditioning equation on parallel architectures could be obtained.

C.5 Approximating the Preconditioning Matrix

C.6 Inverting the Preconditioning Matrix

We develop a parallel, two-level, solver for 3 dimensional (3-D), unstructured grid, nonsymmetric, elliptic problems. The solver is a preconditioned GMRES method with cell centered finite volume spatial discretization. The preconditioner can be viewed as a two-level Schwarz method or a two-grid multigrid V-cycle with aggressive coarsening. Our coarse grid correction employs simple summation and injection for inter-grid transfer operators and a variational method to construct the coarse grid operator. These choices have resulted in a minimum of software complexity.

C.6.1 Introduction

As the field of computational physics matures the algorithmic challenges grow more complex. Among theses challenges are the need to solve large 3-D elliptic problems efficiently, the use of unstructured grids for modeling complex geometries, and the ability to effectively utilize modern parallel computing platforms. In this paper we detail an algorithm for the iterative solution of large 3-D elliptic problems, on unstructured grids, and on modern parallel computing platforms. Our initial motivation for the development of this algorithm has been simulation of casting processes using 3-D unstructured finite volume methods and the development of the TRUCHASsimulation code [64,65]. However, the method is quite general and should find use in a variety of applications such as ground water flow. Also, while we apply this method to stationary unstructured finite volume grids, it could be applied to simulation problems using adaptive mesh refinement.

Our model equations of interest are,

$$\frac{\partial \phi}{\partial t} + \nabla \cdot (-D(\mathbf{r}) \nabla \phi) = s_1(\mathbf{r}), \tag{C.4}$$

and

$$\nabla \cdot (D(\mathbf{r}) \nabla \phi) = s_2(\mathbf{r}). \tag{C.5}$$

In the simulation of casting the first equation arises from heat conduction with phase change and the second equation is a pressure Poisson equation for variable density incompressible flow. The two modern iterative approaches to the solution of linear systems arising from the discretization of these equations are preconditioned Krylov methods [45] and multigrid methods [52]. Paramount to the development of an efficient elliptic solver is the notion of algorithmic scalability. An optimal algorithm is one who's required iteration count is not a function of the grid refinement, and multigrid methods are the most widely understood methods which posses this character. Developing parallel multigrid methods for unstructured grids is more challenging than developing a serial multigrid method for structured grids. Especially if one is required to use a sophiticated diffusion operater on the unstructured grid [66]. A notable success has been the work of Mavriplis [67]. Developing a parallel preconditioned Krylov solver for unstructured grids is rather straight forward. The path of least resistance appears to be using a domain-based preconditioner [68], the simplest being block Jacobi (one-level additive Schwarz). This approach can provide one with good parallel efficiency but it does not possess the property of algorithmic scalability. That is, as the grid is refined, and the number of blocks (or processors) is increased, the number of required Krylov iterations will increase. The two approaches for overcoming this scaling, as put forth in [68], are the addition of a coarse grid correction scheme and overlap between the blocks. In order to minimize the lack of algorithmic scalability we add a simple coarse grid correction scheme to a straight forward block Jacobi preconditioner. In constructing our solver we have attempted to strike a balance between: 1) code complexity and development time, 2) algorithmic scalability, and 3) parallel efficiency.

By "simple coarse grid correction" we mean 3 specific things. The coarsening is predefined by the parallel decomposition of our 3-D grid. This is another way of saying that every processor represents a coarse grid finite volume. Our inter-grid transfer operators are low-complexity, piece-wise constant interpolation. Our coarse grid linear operator is formed in an algebraic manner from our fine grid matrix and our inter-grid transfer operators, i.e. a variational method. As a result of these choices no physics operator is discretized on a coarse grid, and in fact no coarse grid is actually formed.

An additional simplification for unstructured grid problems, afforded us through the use of an outer Krylov iteration, is the fact that we never form the matrix resulting from our true discrete operator. The effect of the true discrete operator is only realized in the matrix-vector multiply of the Krylov method. The only matrix which is formed (for preconditioning purposes) is one which results from a lower-order, more approximate, discrete operator. The true discrete operator is a least squares based method [35, 69] which can in general produce a nonsymmetric matrix. We use the General Minimal Residual (GMRES) algorithm [70] for our Krylov method.

We clearly acknowledge that our method could be described as either a two-grid multigrid V-cycle preconditioner with block Jacobi as the fine grid smoother and a variational coarse space, or as a two-level Schwarz preconditioner which is additive on the fine level, multiplicative between levels and has minimal overlap. This dual distinction has also been acknowledged in [68]. We are less concerned with the exactness of our chosen vernacular and more concerned with the clear presentation of the algorithm and its performance. The number of possible algorithms in the numerical analysis literature which may be viewed as closely related to ours are to numerous to mention. In point of fact, the main algorithmic character of our preconditioner

can be traced back to the early work of Poussin [71], Settari and Aziz [72], and Nicolaides [73].

Finally, we wish to emphasize that this two-level preconditioner can be viewed as a means through which to parallelize one's favorite non-parallel iterative preconditioner such as SSOR, ILU, or algebraic multigrid. In Section 2 we describe our fine grid solver, and Section 3 discusses the coarse grid correction in the preconditioner. Algorithmic performance is given in Section 4, and conclusion is Section 5.

C.6.2 Fine Grid Solver

In this section we outline the details of the fine grid solver which is a preconditioned Krylov method. The preconditioning matrix is constructed using a simplified, approximate, spatial discrete operator. A block Jacobi method is used to iteratively invert this fine grid preconditioner. We use left preconditioning with in the Krylov software package JTpack90 [74] and and utilize the parallel communication library PGSLib [75].

C.6.2.1 Preconditioned Krylov Methods

Probably the best known Krylov method is the conjugate gradient (CG) method made popular in the computational physics community by Kershaw [76]. CG is applicable only to symmetric matrices. Let us compare CG to GMRES [70], which can be applied to nonsymmetric matrices. CG enjoys a short vector recurrence relationship which allows one to construct an orthogonal set of search directions without storing all of the search directions. Thus in CG, work scales linearly, and required storage is constant, as the number of iterations increases. In GMRES, work scales quadratically, and required storage scales linearly, as a function of iteration count. This is because GMRES must store all of the search directions in order to maintain an orthogonal set. An often employed "fix" is to store only k Krylov vectors, GMRES(k). If linear convergence is not achieved after k iterations a new, temporary, linear solution is constructed from the existing k vectors and GMRES is restarted, with this temporary solution as the initial guess. Restarting can significantly effect the convergence rate of GMRES. As a result, when using GMRES as we are here, there is great motivation to keep the required number of GMRES iterations low. We feel this translates into a clear need for effective multilevel preconditioning. We also acknowledge that Krylov methods exist which can be applied to nonsymmetric systems, and which possess storage requirements which are independent of Krylov iteration [77, 78].

It will be our convention to use lower case bold faced letters to represent vectors and upper case bold faced letters to represent matrices. The general linear system arising from the discretization of our model equations is $\mathbf{A}\phi = \mathbf{s}$. The left preconditioned form of is,

$$\mathbf{P}^{-1}\mathbf{A}\phi = \mathbf{P}^{-1}s, \tag{C.6}$$

where $\mathbf{P}$ represents symbolically the preconditioning matrix and $\mathbf{P}^{-1}$ represents its inverse. In practice, this inverse is only approximately realized through some standard iterative process, and the symbol $\tilde{\mathbf{P}}^{-1}$ may be

more appropriate. Each GMRES iteration requires a preconditioned matrix-vector multiply,

$$\mathbf{y} = (\tilde{\mathbf{P}}^{-1}\mathbf{A})\mathbf{v}, \tag{C.7}$$

where $\mathbf{v}$ is the known, n^{th}, search direction and $\mathbf{w}$ represents the first step in forming the $n+1^{st}$ search direction. The multiply requires two steps.
Step 1, matrix vector multiply:

$$\mathbf{w} = \mathbf{A}\mathbf{v}, \tag{C.8}$$

Step 2, Preconditioning (parallel bottle neck):

$$\mathbf{y} = \tilde{\mathbf{P}}^{-1}\mathbf{w}, \tag{C.9}$$

(i.e iteratively solve $\mathbf{P}\mathbf{y} = \mathbf{w}$).

There are two important questions to ask at this point. What approximation of $\mathbf{A}$ to use to form $\mathbf{P}$, and what iterative method to use to approximate $\mathbf{P}^{-1}$. As stated previously our spatial operator on the 3-D unstructured grid is based on a least-squares method [35, 69]. To evaluate $\mathbf{w} = \mathbf{A}\mathbf{v}$ with out evaluating $\mathbf{A}$ is straight forward, but to actually form the elements of $\mathbf{A}$ is more complicated. We chose to form $\mathbf{P}$ from a simple 7-point method for the $\nabla \cdot (-D(\mathbf{r})\nabla)$ operator which assumes a locally orthogonal grid. This not only reduces complexity but also insures a preconditioning matrix which is diagonally dominant. Next we define our approach for approximating $\mathbf{P}^{-1}$.

C.6.2.2 Block Jacobi: Basic Domain Decomposition

The goal of domain-based preconditioners is to decompose the global inversion into a sum of local inversions which can be done on processor, in parallel [68]. The local, on processor, "subdomain" inversions $(\mathbf{P}_i^{sub})^{-1}$ can be direct solves, ILU, SSOR, multigrid or something else. For a 4 subdomain (4 processor) problem we have;

$$\mathbf{P} = \mathbf{P}_1^{sub} + \mathbf{P}_2^{sub} + \mathbf{P}_3^{sub} + \mathbf{P}_4^{sub}. \tag{C.10}$$

Block Jacobi (single pass) assumes;

$$\mathbf{P}^{-1} \approx (\mathbf{P}_1^{sub})^{-1} + (\mathbf{P}_2^{sub})^{-1} + (\mathbf{P}_3^{sub})^{-1} + (\mathbf{P}_4^{sub})^{-1}. \tag{C.11}$$

In [68] This would be referred to as an additive Schwarz method. This approximation degrades with the number of subdomains (i.e processors) and thus the effectiveness of the preconditioner degrades.

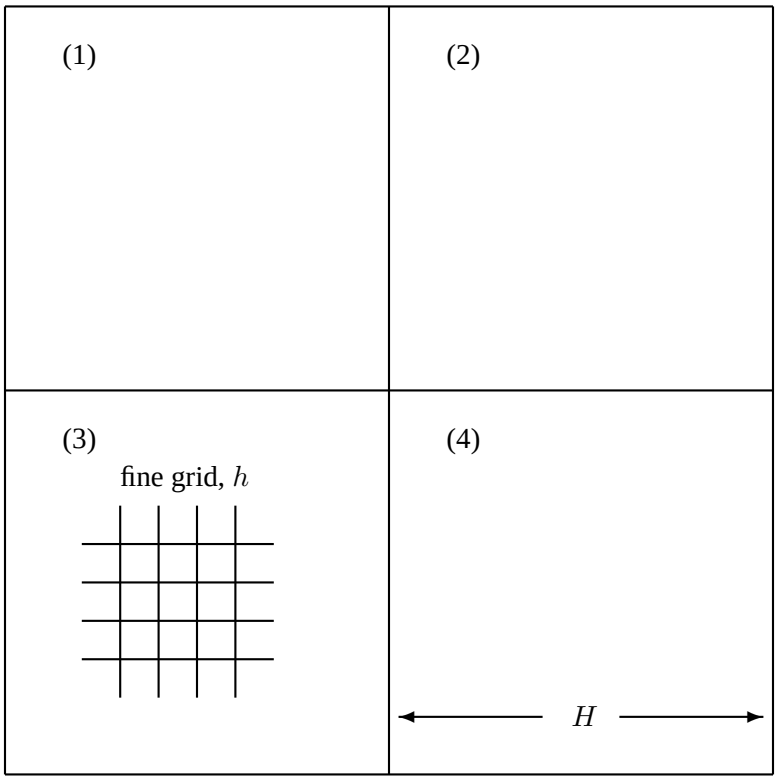


Figure C.1: 4 subdomain example

For further insight consider block Jacobi with 4 subdomains in fig. (C.1). Here the fine grid spatial scale is defined as h, and the subdomain spatial scale is defined as H, with $H \gg h$.

In our parallel implementation the global matrix $\mathbf{P}$ is not stored. The unknowns in each subdomain problem have a unique ordering within that subdomain. This is equivalent to thinking of a global system $\mathbf{Py} = \mathbf{w}$, reordered within subdomains, to look like:

$$\begin{bmatrix} \mathbf{D}_{11} & \mathbf{U}_{12} & \mathbf{U}_{13} & \mathbf{U}_{14} \\ \mathbf{L}_{21} & \mathbf{D}_{22} & \mathbf{U}_{23} & \mathbf{U}_{24} \\ \mathbf{L}_{31} & \mathbf{L}_{32} & \mathbf{D}_{33} & \mathbf{U}_{34} \\ \mathbf{L}_{41} & \mathbf{L}_{42} & \mathbf{L}_{43} & \mathbf{D}_{44} \end{bmatrix} \begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \\ \mathbf{y}_3 \\ \mathbf{y}_4 \end{bmatrix} = \begin{bmatrix} \mathbf{w}_1 \\ \mathbf{w}_2 \\ \mathbf{w}_3 \\ \mathbf{w}_4 \end{bmatrix}.$$

This is a domain-based reordering and results from blocking our physical geometry (finite volumes) into subdomains. Here the block matrix $\mathbf{U}_{12}$ represents the coupling of subdomain 1 to subdomain 2, the block matrix $\mathbf{U}_{13}$ represents the coupling of subdomain 1 to subdomain 3, and so on. The n^{th} block Jacobi iteration, assuming a direct solve on the subdomains, can then be represented by;

$$
\begin{aligned}
\mathbf{y}_1^n &= \mathbf{D}_{11}^{-1}[\mathbf{w}_1 - \mathbf{U}_{12}\mathbf{y}_2^{n-1} - \mathbf{U}_{13}\mathbf{y}_3^{n-1} - \mathbf{U}_{14}\mathbf{y}_4^{n-1}] \\
\mathbf{y}_2^n &= \mathbf{D}_{22}^{-1}[\mathbf{w}_2 - \mathbf{L}_{21}\mathbf{y}_1^{n-1} - \mathbf{U}_{23}\mathbf{y}_3^{n-1} - \mathbf{U}_{24}\mathbf{y}_4^{n-1}] \\
\mathbf{y}_3^n &= \mathbf{D}_{33}^{-1}[\mathbf{w}_3 - \mathbf{L}_{31}\mathbf{y}_1^{n-1} - \mathbf{L}_{32}\mathbf{y}_2^{n-1} - \mathbf{U}_{34}\mathbf{y}_4^{n-1}] \\
\mathbf{y}_4^n &= \mathbf{D}_{44}^{-1}[\mathbf{w}_4 - \mathbf{L}_{41}\mathbf{y}_1^{n-1} - \mathbf{L}_{42}\mathbf{y}_2^{n-1} - \mathbf{L}_{43}\mathbf{y}_3^{n-1}]
\end{aligned} \tag{C.12}
$$

As can be seen this is highly parallel. For $\mathbf{y}^0 = 0$, and one pass, this is equivalent to "block" diagonal scaling, and the $\mathbf{U}$'s and $\mathbf{L}$'s need not be formed. Again, an approximate inverse may be more practical on the subdomains (i.e. replacing $\mathbf{D}_{11}^{-1}$ with $\tilde{\mathbf{D}}_{11}^{-1}$) and $\tilde{\mathbf{D}}_{11}^{-1}$ can be ILU, SSOR, multigrid, or something else.

Since it is our intention to use this block Jacobi method as a fine grid smoother $\mathbf{y}^0$ will not always be zero, and we need to form the $\mathbf{U}$'s and $\mathbf{L}$'s. In our structured grid, 4 subdomain, example forming the $\mathbf{U}$'s and $\mathbf{L}$'s can be thought of as using a 1 cell(finite volume) overlap to form Dirichlet boundary conditions. In our parallel implementation, $\mathbf{D}_{11}, \mathbf{U}_{12}, \mathbf{U}_{13}, \mathbf{U}_{14}$ and $\mathbf{y}_1^n$ reside on processor 1. Copies of the elements of $\mathbf{y}_2^{n-1}, \mathbf{y}_3^{n-1}, \mathbf{y}_4^{n-1}$ which are required to form $\mathbf{y}_1^n$ are also stored on processor 1. This forces some level of inter-processor communication for multiple passes of block Jacobi. Algorithm performance penalties for removing levels of communication will be considered in the future. In the terminology of Schwarz methods [68] this is the extent of our overlap. While it is known that increased overlap improves algorithmic scalability, increased overlap is a challenge for unstructured grids. Thus our fine grid solver is a GMRES method with a block Jacobi preconditioner. The blocks are defined by the part of our geometry which resides on a processor. The local "on processor" solves within the block Jacobi can be direct solves, ILU, SOR, or multigrid. Next we describe our coarse grid correction which is added to the block Jacobi preconditioner.

C.6.3 Coarse Grid Correction Scheme

As stated, our block Jacobi preconditioner, will become less effective with increasing numbers of subdomains / processors, resulting in an undesirable increase in the number of GMRES iterations. Since it is our goal to run on $O(10^3)$ processors we must add something to our algorithm to remedy this scaling. It is well understood that our preconditioner lacks a multilevel component which would enhance global communication. In order to correct this we have opted for a single coarse grid correction scheme where each subdomain represents a "coarse grid finite volume" and thus our coarsening is predefined by our domain decomposition. It will be demonstrated that this simple addition positively impacts the performance of our solver by improving algorithmic scaling without significantly degrading parallel performance.

Consider the following two-level preconditioner which could be referred to as an additive Schwarz method, multiplicative between levels , or as a two-grid V-cycle with a block Jacobi smoother. We desire the iterative solution to $\mathbf{P}\mathbf{y} = \mathbf{w}$ with $f \equiv$ fine, and $c \equiv$ coarse. The act of restriction transfers fine grid data to the coarse grid and the act of prolongation transfers coarse grid data to the fine grid.

1. Relax $\mathbf{y}_f^0$ to $\mathbf{y}_f^1$ by approximately solving $(\mathbf{P}_f)^{-1}(\mathbf{w} - \mathbf{P}_f\mathbf{y}_f^0)$

(Additive Schwarz / block Jacobi is the smoother)

2. Evaluate linear residual $\mathbf{r}es_f = \mathbf{w} - \mathbf{P}_f\mathbf{y}_f^1$, and restrict to coarse grid, $\mathbf{r}es_c = \mathcal{R} * \mathbf{r}es_f$

3. Solve coarse grid problem, $\mathbf{P}_c\delta y_c = \mathbf{r}es_c$, for coarse grid correction δy_c.

4. Prolongate coarse grid correction and update fine grid solution vector, $\mathbf{y}_f^2 = \mathbf{y}_f^1 + \mathcal{P} * \delta y_c$

5. Relax $\mathbf{y}_f^2$ to $\mathbf{y}_f^3$ by approximately solving $(\mathbf{P}_f)^{-1}(\mathbf{w} - \mathbf{P}_f\mathbf{y}_f^2)$
   (Additive Schwarz / block Jacobi is the smoother)

First let us highlight the required inter-processor communication in this preconditioner assuming one pass of block Jacobi at steps 1 and 5. Step 2 requires communication since in terms of our 4 subdomain example $\mathbf{r}es_{f,1}$ is a function of $\mathbf{y}_{f,2}^1$, $\mathbf{y}_{f,3}^1$, and $\mathbf{y}_{f,4}^1$. Step 3 requires communication in order to assimilate or coarse grid operator onto 1 processor where it is solved. Step 4 requires communication to distribute the coarse grid correction to all processors. Step 5 requires communication since our block Jacobi has a nonzero initial guess, $\mathbf{y}_f^2$. This complete level of communication should provide increased robustness and improve algorithmic scaling. Removing levels of communication, and evaluating performance gains or losses, is part of our future plans.

Next we define the inter-grid transfer operators, Restriction and Prolongation ($\mathcal{R}$ and $\mathcal{P}$) and the coarse grid operator $\mathbf{P}_c$. The unstructured grid has motivated the simplest choices for $\mathcal{R}$ and $\mathcal{P}$, summation for $\mathcal{R}$ and injection for $\mathcal{P}$ (piece-wise constant). For $i \equiv$ grid cell index, and $I \equiv$ sub-domain index we have,

$$\mathbf{r}es_c = \mathcal{R} * \mathbf{r}es_f \quad \Rightarrow \quad \mathbf{r}es_{I,c} = \sum_{i \in I} \mathbf{r}es_{i,f}$$

and

$$\mathbf{y}_f^2 = \mathbf{y}_f^1 + \mathcal{P} * \delta\mathbf{y}_c \quad \Rightarrow \quad i \in I, \;\; \mathbf{y}_{f,i}^2 = \mathbf{y}_{f,i}^1 + \delta y_{c,I}$$

Forming the coarse grid operator by re-discetizing the problem on a coarse grid can be challenging with an unstructured grid. It may also involving bringing "physics" down to coarse grid which we view as undesir-

able. We form our coarse grid operator in "Black Box" fashion [79] using our restriction and prolongation operators,

$$\mathbf{P}_c = \mathcal{R} * \mathbf{P}_f * \mathcal{P}. \tag{C.13}$$

The coarse grid operator is constructed algebraically from fine grid operator and we will provide a simple algorithm for this below. This is often referred to as a variational approach and our differs from that of [79] due to the simplicity of our inter-grid transfer operators.

A few comments are in order regarding the simplicity of our inter-grid transfer operators. It is known, within the multigrid community, that to develop an optimal method for a second order PDE either restriction or prolongation must be linear interpolation [52]. We do not satisfy this. However, recent results with multigrid as a preconditioner indicate that the penalty to be paid for this simplicity may not be as significant as when multigrid is a stand alone solver [80, 81]. It should also be noted that if our fine grid operator comes from a conservative finite volume discretization, and we use our simple inter-grid transfer operators to form a coarse grid operator, then our coarse space problem will also be a conservative balance of face fluxes and volume sources. This is known to be a useful property from black box multigrid [79, 82].

As "experimental evidence" that this simple coarse grid correction should produce a positive effect we cite the recent work of Jenssen and Weinerfelt [83] applied to the fully coupled Navier-Stokes equations. Their two-level method is used without a Krylov accelerator, using direct solves on the subdomains, and no post smoothing after the coarse grid correction. They employ summation and injection with finite volumes (identical inter-grid transfer operators to ours), and generate their coarse grid operator with the variational approach.

Finally, we describe the algorithm for constructing the coarse grid operator. We define the total number of finite volumes as N_{fv} and the total number of subdomains as N_{sd}. $\mathbf{P}_f$ is an $N_{fv} \times N_{fv}$ matrix and $\mathbf{P}_c$ is an $N_{sd} \times N_{sd}$ matrix. $\mathcal{P}$ is an $N_{fv} \times N_{sd}$ matrix and $\mathcal{R}$ is an $N_{sd} \times N_{fv}$ matrix. Due to the simplicity of $\mathcal{P}$ and $\mathcal{R}$ they are not formed. We define an element of $\mathbf{P}_f$ as $P_f(i,j)$ and an element of $\mathbf{P}_c$ as $P_c(I,J)$. Given these definitions, and Eq. (C.13), each element of the coarse grid matrix is constructed as

$$P_c(I,J) = \sum_{i \in I} \sum_{j \in J} (P_f(i,j)). \tag{C.14}$$

Each row of $\mathbf{P}_c$ can be constructed on it's own processor from the part of $\mathbf{P}_f$ which lives on that processor. For a clarity, we return to the 4 subdomain example. In this example $P_c(1,1)$ is the sum of all elements in $\mathbf{D}_{1,1}$, $P_c(1,2)$ is the sum of all elements in $\mathbf{U}_{1,2}$, $P_c(1,3)$ is the sum of all elements in $\mathbf{U}_{1,3}$, and $P_c(1,4)$ is the sum of all elements in $\mathbf{U}_{1,4}$. We are not required to have an equal number of finite volumes within each subdomain.

C.6.4 Future Work

- Multiple coarse grid volumes per subdomain
- damped block Jacobi smoother
- reduced and/or altered communication in V-cycle
- more processors

Appendix D

Nonlinear Solution Methods

A continually recurring problem in TRUCHAS is the need to find an approximate solution to

$$F(u) = 0, \tag{D.1}$$

where $F : \Omega \subset \mathbb{R}^m \to \mathbb{R}^m$ is a nonlinear mapping and m is order of the mesh size. Methods of solving (D.1) when F is smooth[1] are typically some variant of Newton's method: if u_k is the current approximate solution, the next approximate solution is $u_{k+1} := u_k + \delta u_{k+1}$, where δu_{k+1} is the solution of the *correction equation*

$$J_k \delta u_{k+1} = -F(u_k) \tag{D.2}$$

with Jacobian $J_k := F'(u_k)$. Unfortunately, it may be very expensive, or even practically impossible, to compute the Jacobian and/or solve the correction equation exactly, especially for larger systems. In such situations, one can instead seek an approximate solution to (D.2), possibly using an approximation for J_k, which gives rise to an *inexact Newton method*. TRUCHAS provides two such methods: the Jacobian-free Newton-Krylov method and an accelerated inexact Newton method.

D.1 Jacobian-Free Newton-Krylov Method

The Generalized Minimal RESidual (GMRES) algorithm [45] is used to solve Equation D.2. GMRES (or any other Krylov method such as conjugate gradients) defines an initial linear residual, $\mathbf{r}_0$ given an initial

[1] Strictly speaking, the nonlinear systems encountered in TRUCHAS may not be smooth everywhere owing, for example, to nonsmoothness in phase diagrams or the solid mechanics contact model. Nevertheless it is useful in in practice to ignore this fact, recognizing that the nonlinear solver convergence behavior will be adversely effected in a neighborhood of the nonsmoothness.

guess, $\delta\mathbf{u}_0$,

$$\mathbf{r}_0 = -\mathbf{F}(\mathbf{u}) - \mathbf{J}\delta\mathbf{u}_0. \tag{D.3}$$

Note that the nonlinear iteration index, k, has been dropped.

The l^{th} GMRES iteration minimizes $\parallel \mathbf{J}\delta\mathbf{u}_l + \mathbf{F}(\mathbf{u}) \parallel_2$ with a least squares approach. $\delta\mathbf{u}_l$ is constructed from a linear combination of the Krylov vectors $\{\mathbf{r}_0, \mathbf{J}\mathbf{r}_0, (\mathbf{J})^2\mathbf{r}_0, ..., (\mathbf{J})^{l-1}\mathbf{r}_0\}$, which were constructed during the previous $l-1$ GMRES iterations. This linear combination of Krylov vectors can be written as,

$$\delta\mathbf{u}_l = \delta\mathbf{u}_0 + \sum_{j=0}^{l-1} \alpha_j (\mathbf{J})^j \mathbf{r}_0, \tag{D.4}$$

where evaluating the scalars α_j is part of the GMRES iteration. Upon examining Equation D.4 we see that GMRES requires the action of the Jacobian only in the form of matrix-vector products, which can be approximated by [84]

$$\mathbf{J}\mathbf{v} \approx \left[\mathbf{F}(\mathbf{u} + \epsilon\mathbf{v}) - \mathbf{F}(\mathbf{u})\right] / \epsilon, \tag{D.5}$$

where $\mathbf{v}$ is a Krylov vector (i.e. one of $\{\mathbf{r}_0, \mathbf{J}\mathbf{r}_0, (\mathbf{J})^2\mathbf{r}_0, ..., (\mathbf{J})^{l-1}\mathbf{r}_0\}$), and ϵ is a small perturbation.

Equation D.5 is a first order Taylor series expansion approximation to the Jacobian, $\mathbf{J}$, times a vector, $\mathbf{v}$. For illustration consider the two coupled nonlinear equations $F_1(u_1, u_2) = 0,\ F_2(u_1, u_2) = 0$. The Jacobian for this problem is;

$$\mathbf{J} = \begin{bmatrix} \frac{\partial F_1}{\partial u_1} & \frac{\partial F_1}{\partial u_2} \\ \\ \frac{\partial F_2}{\partial u_1} & \frac{\partial F_2}{\partial u_2} \end{bmatrix}.$$

Working backwards from Equation D.5, we have;

$$\frac{\mathbf{F}(\mathbf{u} + \epsilon\mathbf{v}) - \mathbf{F}(\mathbf{u})}{\epsilon} = \begin{pmatrix} \frac{1}{\epsilon}\Big[F_1(u_1 + \epsilon v_1, u_2 + \epsilon v_2) - F_1(u_1, u_2)\Big] \\ \\ \frac{1}{\epsilon}\Big[F_2(u_1 + \epsilon v_1, u_2 + \epsilon v_2) - F_2(u_1, u_2)\Big] \end{pmatrix}.$$

Approximating $\mathbf{F}(\mathbf{u} + \epsilon\mathbf{v})$ with a first order Taylor series expansion about $\mathbf{u}$, we have;

$$\frac{\mathbf{F}(\mathbf{u}+\epsilon\mathbf{v})-\mathbf{F}(\mathbf{u})}{\epsilon} \approx \begin{pmatrix} \frac{1}{\epsilon}\Big[F_1(u_1,u_2)+\epsilon v_1\frac{\partial F_1}{\partial u_1}+\epsilon v_2\frac{\partial F_1}{\partial u_2}-F_1(u_1,u_2)\Big] \\ \\ \frac{1}{\epsilon}\Big[F_2(u_1,u_2)+\epsilon v_1\frac{\partial F_2}{\partial u_1}+\epsilon v_2\frac{\partial F_2}{\partial u_2}-F_2(u_1,u_2)\Big] \end{pmatrix}.$$

This expression can be simplified to;

$$\begin{pmatrix} v_1\frac{\partial F_1}{\partial u_1}+v_2\frac{\partial F_1}{\partial u_2} \\ \\ v_1\frac{\partial F_2}{\partial u_1}+v_2\frac{\partial F_2}{\partial u_2} \end{pmatrix} = \mathbf{J}v$$

This matrix-free approach, besides its obvious memory advantage, has many unique capabilities. Namely, Newton-like nonlinear convergence without *forming* or *inverting* the true Jacobian.

To complete the description of this technique we provide a prescription for evaluating the scalar perturbation. In this study ϵ is given by,

$$\epsilon = \frac{1}{N||\mathbf{v}||_2}\sum_{m=1}^{N} b|u_m|, \tag{D.6}$$

where N is the linear system dimension and b is a constant whose magnitude is approximately the square root of machine roundoff ($b = 10^{-5}$ for most of this study).

We employ right preconditioning and thus we are solving,

$$(\mathbf{J}P^{-1})(\mathbf{P}\delta\mathbf{u}) = -\mathbf{F}(\mathbf{u}). \tag{D.7}$$

$\mathbf{P}$ symbolically represents the preconditioning matrix and $\mathbf{P}^{-1}$ the inverse of preconditioning matrix. In practice, this inverse is only approximately realized through some standard iterative method, and thus we may think of it more as $\tilde{\mathbf{P}}^{-1}$.

The right preconditioned matrix-free algorithm is:

$$\mathbf{J}\tilde{\mathbf{P}}^{-1}\mathbf{v} \approx [\mathbf{F}(\mathbf{u}+\epsilon\tilde{\mathbf{P}}^{-1}\mathbf{v})-\mathbf{F}(\mathbf{u})] \;/\; \epsilon, \tag{D.8}$$

This is done is actually done in two steps;

1. Solve (iteratively, and not to convergence) $\mathbf{P}y = \mathbf{v}$ for $\mathbf{y}$
2. Perform $\mathbf{Jy} \approx [\mathbf{F}(\mathbf{u} + \epsilon\mathbf{y}) - \mathbf{F}(\mathbf{u})] \ / \ \epsilon$,

Thus only the matrix $\mathbf{P}$ is formed and only the matrix $\mathbf{P}$ is iteratively inverted.

There are two choices to be made;

1. What linearization should be used to form $\mathbf{P}$?
2. What linear iterative method should be used to solve $\mathbf{P}y = \mathbf{v}$?

D.2 An Accelerated Inexact Newton Method

In this section we consider a nonlinear Krylov acceleration procedure for inexact Newton's method that was introduced in [85] (see also [86]). This method falls into a broad class of accelerated inexact Newton (AIN) schemes discussed in [87], though it differs from the specific methods described there.

In the inexact Newton method the correction equation (D.2) is solved only approximately, and often with an approximation M_k for the Jacobian J_k as well. Formally we may express this inexact correction v_{k+1} as

$$v_{k+1} := -P(u_k)^{-1}F(u_k), \tag{D.9}$$

where $P(u_k)^{-1}$ is the *preconditioner* for the nonlinear system. Commonly P^{-1} is only realized as an iterative procedure applied to the system $M_k v = -F(u_k)$; neither P nor P^{-1} need ever be explicitly formed. The ideal preconditioner would be the exact inversion of J_k, yielding Newton's method.

Defining $G(u) = -P(u)^{-1}F(u)$, we recognize that this inexact Newton iteration is simply the standard fixed point iteration $u_{k+1} := u_k + G(u_k)$, which converges if G' is sufficiently close to $-I$ in a neighborhood of the root. Indeed, if $P(u) = F'(u)$ then $G' = -I$ at the root.

With this fixed point iteration view of inexact Newton as a reference, the accelerated method proceeds with successive approximations $u_{k+1} = u_k + v_{k+1}$, $k = 0, 1, \ldots$ At iteration k we have accumulated from previous iterations the correction space $\mathcal{V}_k = \text{span}\{v_1, \ldots, v_k\}$ and corresponding G-difference space $\mathcal{W}_k = \text{span}\{w_1, \ldots, w_k\}$, where $w_n = G(u_n) - G(u_{n-1})$. The fixed point iteration chooses $v_{k+1} = G(u_k)$ as the next correction. The ideal correction would solve $G(u_k + v_{k+1}) = 0$, which isn't feasible, but linearizing one could choose v_{k+1} such that

$$0 = G(u_k) + G'(u_k)v_{k+1}. \tag{D.10}$$

The fixed point iteration, not knowing anything about G', invokes the assumption that $G' \approx -I$ and so substitutes $-I$ for G'. However, if G' is nearly constant in a neighborhood of the root then

$$w_n \approx G' v_n \tag{D.11}$$

so that one does (approximately) know G' on the space $\mathcal{V}_k$. Hence we split the correction in (D.10) into two pieces: $v_{k+1} = p + q$ with $p \in \mathcal{V}_k$, for which we know the action of G' by (D.11), and leftover piece q, for which we do not and so will replace G' by $-I$ as in the fixed point iteration. Introducing the matrices $V_k = [v_1 \dots v_k]$ and $W_k = [w_1 \dots w_k]$, this split correction is

$$p = V_k y, \quad 0 = G(u_k) + W_k y - q, \tag{D.12}$$

where $y \in \mathbb{R}^k$ is chosen so that $W_k y$ is the best l_2 fit to $-G(u_k)$. The accelerated correction is thus

$$v_{k+1} := G(u_k) - (V_k + W_k)(W_k^T W_k)^{-1} W_k^T G(u_k). \tag{D.13}$$

For the first iteration ($k = 0$) the spaces $\mathcal{V}_0$ and $\mathcal{W}_0$ are empty, and the correction v_1 is identical to the fixed point iteration correction $G(u_0)$.

Notice that there is nothing that requires one to use *all* the previous corrections $v_1, \dots, v_k$ (and their corresponding G-differences) when computing the accelerated correction (D.13). In fact, since G is nonlinear, we might expect only the most recent corrections to yield decently accurate information about G'. Also there is nothing to guarantee that the w_n vectors are linearly independent. The accelerated method, therefore, assembles the pair of acceleration spaces $(\mathcal{V}_k, \mathcal{W}_k)$ by taking the vectors in reverse order, starting with (v_k, w_k), and dropping any pair (v_n, w_n) whenever w_n is nearly in the span of the preceding w-vectors. Moreover, the number of vectors used is limited to a maximum number, which is typically very small so that the cost, in time and memory, to compute the accelerated correction is quite modest.

D.3 Preconditioning

Appendix E

Sensitivity Analysis

Sensitivity analysis plays a vital role in trade-off, reliability, inverse/identification, and optimization studies. For example, in a trade-off study one can view the temperature sensitivity field to determine where the most drastic temperature changes will occur if one alters the value of a prescribed Neumann temperature boundary condition.

By definition, the sensitivities are the derivatives of the system response with respect to the model parameters. To compute the sensitivities in TRUCHASthe direct differentiation and semi-analytical methods are combined resulting in efficient and accurate computations that require minimal code modifications.

To explain this sensitivity analysis, consider an analysis which is descretized into a series of cell equations and then assembled into a residual vector (cf. equation D.1) $\mathbf{F}(\mathbf{u})$ where $\mathbf{u}$ is the discretized state vector. The nonlinear equation $\mathbf{F}(\mathbf{u}) = \mathbf{0}$ is then solved for $\mathbf{u}$ in the *primal analysis* by using the nonlinear solver discussed in Appendix D. Of course, if we change a model parameter s, then the residual and hence the state change. To reflect this dependency, the residual equation is expressed as

$$\mathbf{F}(\mathbf{u}(s), s) = \mathbf{0} \tag{E.1}$$

where it is understood that the parameter s is a known input. To determine the sensitivity of the state with respect to a parameter s, one could perturb the parameter by the amount δ, i.e. set $s \longrightarrow s + \delta$, solve the residual equation

$$\mathbf{F}(\mathbf{u}(s + \delta), s + \delta) = \mathbf{0} \tag{E.2}$$

and compute the derivative from the approximation

$$\mathbf{u}'(s) \approx \frac{1}{\delta} \left[\mathbf{u}(s + \delta) - \mathbf{u}(s)\right] \tag{E.3}$$

This *finite difference* sensitivity analysis is simple to implement as the analysis is merely repeated with the perturbed parameter. However, it is computationally unattractive since it requires an additional nonlinear

analysis for each parameter s, for which the sensitivities are desired. Moreover, it is susceptible to truncation errors if δ is too large and round-off errors if δ is too small. Of course the truncation error could be reduced by using a second-order accurate approximation, i.e.

$$\mathbf{u}'(s) \approx \frac{1}{2\,\delta}\left[\mathbf{u}(s+\delta) - \mathbf{u}(s-\delta)\right] \tag{E.4}$$

however, this would require another nonlinear analysis to evaluate the perturbed state $\mathbf{u}(s-\delta)$.

Two methods, the *adjoint* and *direct differentiation*, have been used to efficiently and accurately evaluate sensitivities for a wide variety of systems. The direct approach has been adopted in TRUCHASto avoid the cumbersome backward time mappings that are required by the adjoint method [88]. Due to the transient nature of the system it is necessary to rewrite the residual equation as

$$\mathbf{F}(\mathbf{u}_n(s), \mathbf{u}_{n-1}(s), s) = \mathbf{0} \tag{E.5}$$

In the above equation, the state $\mathbf{u}_n(s)$ evaluated at time $t = t_n$ is determined in the primal analysis given the known inputs: the state $\mathbf{u}_{n-1}(s)$ at time previous time step $t = t_{n-1}$ and the parameter s. Of course, $\mathbf{u}_{n-1}(s)$ is known as it is evaluated during the primal analysis at the previous time step and for the initial time, i.e. $t = t_0$, $\mathbf{u}_0(s)$ is the known initial condition. And, of course $\mathbf{u}_{n-1}(s)$ is a function of s as it changes if s changes.

Differentiating Equation E.5 with respect to s and rearranging gives the *pseudo problem*

$$\mathbf{J}\,\mathbf{u}'_n = -\left\{\frac{\partial \mathbf{F}}{\partial \mathbf{u}_{n-1}}\,\mathbf{u}'_{n-1} + \frac{\partial \mathbf{F}}{\partial s}\right\} \tag{E.6}$$

where $\mathbf{J}$ is the Jacobian of Equation D.2. The *pseudo load*, i.e. the right-had side of Equation E.6 is known since we know the dependency of the residual on $\mathbf{u}_{n-1}$ and s and since the sensitivity $\mathbf{u}'_{n-1}(s)$ is evaluated during the pseudo analysis at the previous time step and since for the initial time $\mathbf{u}'_0(s)$ is the known initial condition sensitivity.

Note that the pseudo problem is linear in the sensitivity $\mathbf{u}'_n(s)$ so the computation of the sensitivity $\mathbf{u}'_n(s)$ requires a linear solve even though the computation of the state $\mathbf{u}_n(s)$ requires a nonlinear solve. In total, one linear pseudo problem is solved for each parameter s for which the sensitivity is desired.

Also note that the linear solve uses the same coefficient matrix that appears in Equation D.2. Thus, the same solver that is used to resolve Equation D.2 can be used to resolve Equation E.6. In the best case scenario, the Jacobian $\mathbf{J}$ is assembled and factored so that the solution of the each pseudo problem (one pseudo problem is solved for each design parameter) only requires a back solve. However, this is seldom the case. So in an effort to hasten the pseudo analyses, one may wish to invest extra effort to develop an effective preconditioner for $\mathbf{J}$ during the primal analysis so it can also be utilized for the subsequent pseudo analyses.

The challenge of the sensitivity analysis is the ability to evaluate the pseudo load, i.e. the right-hand side of Equation E.6. The derivatives can be computed analytically or by symbolic differentiation software.

However, in TRUCHASthe *semi-analytical* method is used whereby the pseudo load is approximated from the second-order accurate finite difference approximation

$$\begin{aligned}\left\{\frac{\partial \mathbf{F}}{\partial \mathbf{u}_{n-1}}\,\mathbf{u}'_{n-1}+\frac{\partial \mathbf{F}}{\partial s}\right\}\Bigg|_{(\mathbf{u}_n,\mathbf{u}_{n-1},s)=(\mathbf{u}_n(s),\mathbf{u}_{n-1}(s),s)} \\ \approx \frac{1}{2\,\delta}\left\{\left[\mathbf{F}(\mathbf{u}_n(s),\mathbf{u}_{n-1}(s)+\delta\,\mathbf{u}'_{n-1}(s),s)+\mathbf{F}(\mathbf{u}_n(s),\mathbf{u}_{n-1}(s),s+\delta)\right]-\right. \\ \left.\left[\mathbf{F}(\mathbf{u}_n(s),\mathbf{u}_{n-1}(s)-\delta\,\mathbf{u}'_{n-1}(s),s)+\mathbf{F}(\mathbf{u}_n(s),\mathbf{u}_{n-1}(s),s-\delta)\right]\right\}\end{aligned} \tag{E.7}$$

Combining Equation E.6 and Equation E.7 gives the pseudo problem

$$\begin{aligned}\mathbf{J}\,\mathbf{u}'_n \;=\; & \frac{1}{2\,\delta}\left\{\left[\mathbf{F}(\mathbf{u}_n(s),\mathbf{u}_{n-1}(s)+\delta\,\mathbf{u}'_{n-1}(s),s)+\mathbf{F}(\mathbf{u}_n(s),\mathbf{u}_{n-1}(s),s+\delta)\right]-\right. \\ & \left.\left[\mathbf{F}(\mathbf{u}_n(s),\mathbf{u}_{n-1}(s)-\delta\,\mathbf{u}'_{n-1}(s),s)+\mathbf{F}(\mathbf{u}_n(s),\mathbf{u}_{n-1}(s),s-\delta)\right]\right\}\end{aligned} \tag{E.8}$$

Several variables must be carefully selected to accurately and efficiently compute the sensitivities. As previously mentioned, improper choices of δ can lead to round-off or truncation errors so the `Sens_Variable--Sens_Variable_Pert` variable must be carefully prescribed. To reduce computation for the pseudo load evaluation a forward difference approximation can be used in place of the default central difference approximation via the `NUMERICS--Energy_Sensitivity_Pseudo_Load` variable. Naturally the forward difference approximation will also result in decreased accuracy. Because the sensitivity analysis assumes that $\mathbf{F}(\mathbf{u}_n(s),\mathbf{u}_{n-1}(s),s)=\mathbf{0}$, cf. Equation E.5, the convergence criterion for the nonlinear solver, i.e. the `NONLINEAR_SOLVER--Convergence_Criterion` (of the `NUMERICS-- Energy_Nonlinear_Solution` namelist), variable, must also be carefully prescribed (of course, this is good practice to ensure accurate computation of the state $\mathbf{u}$). Finally, the ability to accurately and efficiently solve the linear pseudo problem of Equation E.8 requires careful consideration of the `LINEAR_SOLVER-- Convergence_Criterion, Maximum_Iterations, Preconditioning_Method, Preconditioning_Preconditioner, Preconditioning_Scope, Preconditioning_Steps, Relaxation_Parameter, and Stopping_Criterion` (of the `NUMERICS-- Energy_Sensitivity_Solution` namelist) variables.

Appendix F

Tensor Product Mesh Generation

A *tensor product* mesh is characterized by cells that are orthogonal quadrilaterals (rectangles) in 2-D or orthogonal hexahedra (bricks) in 3-D. In addition, the edges of each (hex or quad) cell in a tensor product mesh possess lengths that are in general different. A *uniform* tensor product mesh is one in which all cell edge lengths are identical. While a uniform mesh is attractive from a numerical discretization point of view (e.g., it is easier to quantify discretization errors), it is frequently more computationally efficient to use a nonuniform mesh so that cells are concentrated in a particular region interest, such as in an area where solution gradients are steeper. Although there are several methods for generating a nonuniform mesh, described herein is a method which is preferred in the TELLURIDEproject – and implemented in the TRUCHAScode – namely a mesh in which the ratios of widths or any pairs of adjacent cells is a constant [89]. This type of mesh, described below, is term a "ratio-zoned" mesh.

F.1 Description of a 1-D Ratio-Zoned Mesh

Assume an interval of width w must be descretized,

$$X_0 \leq X \leq X_0 + w\,, \tag{F.1}$$

with a "stretched" grid of N cells having the property

$$\frac{X_{i+1} - X_i}{X_i - X_{i-1}} = \beta\,, \tag{F.2}$$

for $1 < i < N$. The above equation can be written in continuous form as

$$\frac{dX}{di} = a\,\beta^i\,, \qquad 0 \leq i \leq N\,, \tag{F.3}$$

where a is a constant. Integrating Equation F.3:

$$\int_{X_0}^{X} dX = \int_0^i a\beta^i \, di \,, \tag{F.4}$$

gives

$$X_i - X_0 = \frac{a}{\ln \beta} \left(\beta^i - 1\right) \,. \tag{F.5}$$

To simplify the discussion, assume $X_0 = 0$ and divide Equation F.5 by w to give

$$x_i = \frac{a}{\ln \beta} \left(\beta^i - 1\right) \,, \tag{F.6}$$

where $x = X/w$ and a is now a dimensionless constant. The location of the mesh intervals, x_i, is given in Equation F.6 as a function of three variables: a, β, and i. Two auxiliary equations are further required to solve Equation F.6. Given the problem definition, we have the first:

$$x(a, \beta, i = N) = 1 \,; \tag{F.7}$$

and, with loss of generality, the width of the first cell (Δ) is assumed to be known, giving the second:

$$x(a, \beta, i = 1) = \Delta \,. \tag{F.8}$$

Equation F.6, subject to the requirement of Equation F.8, can be solved for a:

$$a = \frac{\Delta \ln \beta}{\beta - 1} \,; \tag{F.9}$$

which allows Equation F.6 to be written as

$$x_i = \frac{\Delta}{\beta - 1} \left(\beta^i - 1\right) \,, \qquad 0 \le i \le N \,. \tag{F.10}$$

Equation F.10, which is the formula for the sum of the first i terms of a geometric series, may be written as

$$f = \frac{\Delta}{\beta - 1} \left(\beta^N - 1\right) - 1 = 0 \,. \tag{F.11}$$

using the requirement of Equation F.7. The method used to solve Equation F.11 depends upon whether β or N is known. Each case is considered in the following.

F.1.1 Case 1: N is Given; Find β

Because $f(\beta)$ is a monotonic function (for $\beta \ge 0$, $N > 0$), Equation F.11 may be readily solved for β using a Newton-Raphson (NR) method:

$$\beta_{k+1} = \beta_k - \frac{f_k}{f'_k} \,, \tag{F.12}$$

where

$$f' = \frac{\partial f}{\partial \beta} = \frac{\Delta}{\beta - 1}\left[N\beta^{N-1} - \frac{\beta^N - 1}{\beta - 1}\right] \tag{F.13}$$

and k is the iteration count.

F.1.2 Case 2: β is Given; Find N

If β is known, Equation F.11 may be solved directly for N:

$$N^* = \frac{\ln\left(1 + \frac{\beta - 1}{\Delta}\right)}{\ln \beta}\,. \tag{F.14}$$

The value of N^* given by Equation F.14 is in general not an integer, so N is found by rounding N^* up to the next whole number:

$$N = \texttt{int}(N^* + 1)\,. \tag{F.15}$$

F.1.3 Bounds for β

Although $f(\beta)$ is a monotonic function, the magnitude of f and its derivatives can be large in the neighborhood of the root if the solution to Equation F.11 is close to unity. This can cause difficulties for the iteration algorithm (Equation F.12) unless a suitable initial guess, β_0, is given.

If $\beta < 1$, then the size of the last cell will be smaller than the average cell size:

$$\Delta\beta^{N-1} < 1/N\,, \qquad \text{or} \qquad \beta < \left(\frac{1}{\Delta N}\right)^{1/N-1}\,. \tag{F.16}$$

A lower bound for β (when $\beta < 1$) can be found by allowing N to become unbounded in Equation F.11:

$$\lim_{N\to\infty} \frac{\Delta}{\beta_{\min} - 1}\left[\beta_{\min}^N - 1\right] - 1\,, \qquad \text{or} \qquad \beta_{\min} = 1 - \Delta\,. \tag{F.17}$$

Thus N may be computed from Equations F.14 and F.15 provided $\beta > \beta_{\min}$.

If $\beta > 1$, the size of the last cell will be larger than the average cell size:

$$\Delta\beta^{N-1} > 1/N\,, \qquad \text{or} \qquad \beta > \left(\frac{1}{\Delta N}\right)^{1/N-1}\,. \tag{F.18}$$

An upper bound for the $\beta > 1$ case follows by observing that the sum of the widths of the first $N-1$ cells is larger than $N-1$ times the smallest cell size:

$$1 - \Delta\beta^{N-1} > (N-1)\Delta \,, \text{or} \quad \beta < \left(\frac{1-(N-1)\Delta}{\Delta}\right)^{1/N-1} . \tag{F.19}$$

Bounds for β, given in Equations F.16 through F.19, may be summarized as

$$1 - \Delta < \beta < \left[\frac{1}{\Delta}\right]^{1/N-1} \text{for} \quad \beta < 1 \,, \tag{F.20}$$

and

$$\left[\frac{1}{\Delta N}\right]^{1/N-1} < \beta < \left[\frac{1-(N-1)\Delta}{\Delta}\right]^{1/N-1} \text{for} \quad \beta > 1 \,. \tag{F.21}$$

F.2 Parameterizing the 1-D Ratio-Zoned Mesh

Six parameters characterize the 1-D ratio-zoned tensor product mesh in the previous section:

- β - the ratio of widths of any pair of adjacent cells;
- Δ_{L} - the first cell width in the mesh interval;
- Δ_{R} - the last cell width in the mesh interval;
- N - the number of cells in the mesh interval;
- w - the total width of the mesh interval; and
- a - the mesh constant given by Equation F.9.

A simple program can be written to compute three out of the six parameters above, with the other three having been specified. Six combinations of of user-specified (known) input and computed (unknown) output are possible:

1. Given Δ_{L}, N, and w; compute Δ_{R}, a, and β.
2. Given Δ_{R}, N, and w; compute Δ_{L}, a, and β.
3. Given Δ_{L}, β, and w; compute Δ_{R}, a, and N.

4. Given Δ_{R}, β, and w; compute Δ_{L}, a, and N.
5. Given Δ_{L}, β, and N; compute Δ_{R}, a, and w.
6. Given Δ_{R}, β, and N; compute Δ_{L}, a, and w.

To insure that the iteration procedure in Equation F.12 converges, β_0 is chosen to be the upper bound from Equation F.20 or Equation F.21. When solving for N^* in Equation F.14, the program uses $\beta = 1.01\beta_{\min}$ if the input value of β is less than $\beta_{\min}$.

F.3 Summary

An interval of width w is meshed with N cells using an exponential function:

$$X_i = \frac{\Delta_{\mathrm{L}}(\beta^i - 1)}{\beta - 1} \quad \text{or} \quad X_i = w - \frac{\Delta_{\mathrm{R}}(\alpha^{N-i} - 1)}{\alpha - 1} \tag{F.22}$$

where $0 \le i \le N$, $0 \le X \le w$, and

$$\begin{aligned} X_i &= \text{coordinate of grid point } i\,, \\ w &= \text{width of the mesh interval}\,, \\ N &= \text{number of cells in the mesh interval}\,, \\ \beta &= \text{ratio} = (X_{i+1} - X_i)/(X_i - X_{i-1})\,, \\ \alpha &= 1/\beta, \\ \Delta_{\mathrm{L}} &= \text{width of left (first) cell} = X_1 - X_0\,, \\ \Delta_{\mathrm{R}} &= \text{width of right (last) cell} = X_N - X_{N-1}\,. \end{aligned} \tag{F.23}$$

Setting $i = N$ and $i = 0$ in Equation F.22 gives

$$X_N = w = \frac{\Delta_{\mathrm{L}}(\beta^N - 1)}{\beta - 1} = \frac{\Delta_{\mathrm{R}}(\alpha^N - 1)}{\alpha - 1}\,. \tag{F.24}$$

Setting $i = N - 1$ in Equation F.22 gives

$$X_{N-1} = \frac{\Delta_{\mathrm{L}}(\beta^{N-1} - 1)}{\beta - 1}\,, \tag{F.25}$$

which, when subtracted from Equation F.24, gives

$$\Delta_{\mathrm{R}} = \Delta_{\mathrm{L}}\beta^{N-1}\,. \tag{F.26}$$

When any three variables of the set $(\beta, N, \Delta_{\mathrm{R}}, \Delta_{\mathrm{L}}, w)$ are known, Equations F.24 and F.26 may be used to find the unknown variables.

F.4 Specifying a Tensor Product Mesh for TRUCHAS

In TRUCHAS, the current set of allowed input parameters are β, N, and w, which gives simple algebraic expressions for Δ_{L} and Δ_{R}, namely:

$$\Delta_{\mathrm{L}} = w\frac{\beta - 1}{\beta^N - 1} \quad \text{and} \quad \Delta_{\mathrm{R}} = \Delta_{\mathrm{L}}\beta^{N-1} = w(\beta - 1)\,. \tag{F.27}$$

Appendix G

Volume Fraction Generation

Appendix H

Plane Truncation of Hexahedral Volumes

Appendix I

Grid Mapping

The following chapter presents the TRUCHAS capability for mapping cell quantities between meshes.

I.1 Introduction

When performing a computational physics simulation in a given physical domain, it may be that two independent unstructured grids are defined on this domain and it may be necessary to map quantities between these meshes.

One situation occurs when "multi-physics" is present. That is, if there are multiple meshes in use for simulation of different kinds of physical processes occuring at the same time, it may be necessary to map quantities between the various meshes. This case is encountered in the Truchas casting simulation, where a simplicial (i.e., tetrahedral) mesh is required to simulate electromagnetism with the resulting computed inductive heating quantities (defined on the simplicial mesh) having to be mapped in a conservative fashion onto an unstructured hexahedral mesh that is used for computation of heat-transfer, phase change, and thermomechanical effects.

Another situation occurs during "restarts". If a physics simulation is run over a time interval (say $[T_1, T_2]$), it may have to be restarted on a different mesh in order to simulate the next time interval of interest (i.e., $[T_2, T_3]$). It may be that the mesh was deformed over the first time interval and was no longer usable, or that the physical phenomena occuring in the two time intervals are different enough in character that a new mesh is called for. This again happens in the Truchas code where a temperature field is passed between subsequent stages of simulation which describe different stages of the casting process.

Previous research has involved mappings between a 3-D unstructured mesh and a 3-D Cartesian mesh [90]

which is from an algorithmic perspective a much simpler task than the task of mapping between two fully 3-D unstructured meshes. In [91], techniques are given for mapping quantities between a pair of unstructured 2-D surface meshes. An excellent discussion is given in that reference of the quandary that arises when mesh geometries do not match exactly and it is impossible to both preserve constant fields and be exactly conservative in the mapping process.

In this chapter, we present an algorithm for rapidly and accurately mapping cell-based quantities from a source mesh potentially consisting of hexahedra ("hexes"), prisms, pyramids, or tetrahedra ("tets") to a destination mesh occupying the same domain, which also consists of hexes, prisms, pyramids, or tets.

I.2 Theory

Our task is to map a cell-based function defined on unstructured elements of 'Mesh A' to a cell-based function defined on unstructured 'Mesh B' quickly and in a conservative or near-conservative fashion. We assume the elements of both meshes come from the usual 'zoo' of 3-D elements: hexahedra, prisms, pyramids, and tetrahedra. Let $f^B(\mathbf{x})$ be the function defined by the n_B cell values for the B mesh and $f^A(\mathbf{x})$ be our function defined by the n_A cell values for the A mesh. Here $\mathbf{x} \in \Omega_B = \bigcup_{i=1}^{n_B} B_i$, the computational domain defined by the elements of the mesh B. This domain is assumed to be face-connected. I.e., we assume that between any two elements in mesh B, say $B_\alpha$and B_β, there is a path

$$B_\alpha = B_{i_1}, B_{i_2}, B_{i_3}, \ldots, B_{i_m} = B_\beta$$

such that B_{i_k} and $B_{i_{k+1}}$ share a (triangular or quadrilateral) face for all $1 \leq k \leq m-1$. Similarly, we assume the function $f^A(\mathbf{x})$ is defined on a domain $\Omega_A = \bigcup_{j=1}^{n_A} A_j$ which is the union of face-connected elements that comprise mesh A. For purposes of our mapping algorithm, we assume $\Omega_A \subseteq \Omega_B$. This condition will be slightly relaxed later.

Let f_i^B be the cell values of f^B on the mesh B, and f_j^A be the cell values of f^A on the mesh A. Let $\chi_{_{B_i}}$ be the characteristic function for element B_i which is defined by

$$\chi_{_{B_i}}(\mathbf{x}) = \begin{cases} 1, & \mathbf{x} \in B_i \\ 0, & \mathbf{x} \notin B_i \end{cases} .$$

Similarly, let $\chi_{_{A_j}}$ be the characteristic function for element A_j. In this notation, $f^B(\mathbf{x}) = \sum_i f_i^B \chi_{_{B_i}}(\mathbf{x})$ and $f^A(\mathbf{x}) = \sum_j f_j^A \chi_{_{A_j}}(\mathbf{x})$. (This conveniently defines $f^B(\mathbf{x}) \equiv 0,\ \mathbf{x} \notin \Omega_B$ and $f^A(\mathbf{x}) \equiv 0,\ \mathbf{x} \notin \Omega_A$.) For our map to be exactly conservative,

$$\int_{\Omega_B} f^B(\mathbf{x})\, dV = \int_{\Omega_A} f^A(\mathbf{x})\, dV. \tag{I.1}$$

Obviously if $f^B(\mathbf{x}) = f^A(\mathbf{x})\ \forall \mathbf{x}$, this would be satisfied. Of course, this is not possible in general, but since we have n_B undetermined cell values f_i^B to work with, we can force $f^B = f^A$ in the "weak sense"

by requiring

$$\int_{\Omega_B} f^B \chi_{_{B_i}}\, dV = \int_{\Omega_A} f^A \chi_{_{B_i}}\, dV, \qquad 1 \le i \le n_B. \tag{I.2}$$

These n_B equations will fix the n_B unknown coefficients f_i^B. Indeed from $(I.2)$ we have

$$\int_{\Omega_B} \sum_k f_k^B \chi_{_{B_k}} \chi_{_{B_i}}\, dV = \int_{\Omega_A} \sum_j f_j^A \chi_{_{A_j}} \chi_{_{B_i}}\, dV.$$

So

$$|B_i| f_i^B = \sum_j |B_i \cap A_j| f_j^A,$$

where $|B_i|$ denotes the volume of B_i and $|B_i \cap A_j|$ denotes the volume of the intersection of element B_i with element A_j. This implies that we should set

$$f_i^B = \frac{1}{|B_i|} \sum_j |B_i \cap A_j| f_j^A. \tag{I.3}$$

Thus, with this choice for f_i^B, we have (I.2) obeyed $\forall B_i$, and summing over $1 \le i \le n_B$, we obtain the desired conservation condition (I.1).

Notice now that if $\Omega_A = \Omega_B$, then $\sum_j |B_i \cap A_j| = |B_i|$, so that (I.3) defines f_i^B as a *volume-weighted average of the* f_j^A. This weighted average has two desirable properties: (i) it is *local*, so that the f_i^B do not unphysically depend on f_j^A values that are not in the neighborhood of B_i. (ii) it is an average with non-negative weights, so that no new unphysical maxima and minima are created in any neighborhood. That is,

$$\min_{A_j \cap B_i \neq \emptyset} f_j^A \le f_i^B \le \max_{A_j \cap B_i \neq \emptyset} f_j^A, \quad 1 \le i \le n_B. \tag{I.4}$$

In practice, our algorithm often only computes approximate values for the intersection

$$V_{ij} \approx |B_i \cap A_j|. \tag{I.5}$$

We have three choices then. First, we could use (I.3) and perform the replacement (I.5) to obtain

$$f_i^B = \frac{1}{|B_i|} \sum_j V_{ij} f_j^A. \tag{I.6}$$

This produces a cell field $\{f_i^B\}$ that is neither conservatively mapped (I.1) or a weighted average, but is what we call a "raw" mapping. A second choice is to enforce conservation by multiplying the V_{ij} by a suitable factor. In fact, if we replace V_{ij} by $V_{ij} \frac{|A_j|}{\sum_m V_{mj}}$ in (I.6), to obtain

$$f_i^B = \frac{1}{|B_i|} \sum_j V_{ij} \frac{|A_j|}{\sum_m V_{mj}} f_j^A, \tag{I.7}$$

then we have that conservation has been restored:

$$\begin{aligned}
\int_{\Omega_B} f^B \, dV &= \sum_i |B_i| f_i^B \\
&= \sum_i |B_i| \left(\frac{1}{|B_i|} \sum_j V_{ij} \frac{|A_j|}{\sum_m V_{mj}} f_j^A \right) \\
&= \sum_j |A_j| \frac{\sum_i V_{ij}}{\sum_m V_{mj}} f_j^A \\
&= \int_{\Omega_A} f^A \, dV.
\end{aligned}$$

The third option is to keep our mapping's "weighted average" property that preserves maxima and minima. Thus we replace (I.6) by

$$f_i^B = \frac{1}{\sum_k V_{ik}} \sum_j V_{ij} f_j^A. \tag{I.8}$$

With this choice, assuming that our approximate V_{ij} are all non-negative and are only positive for those j where the corresponding A_j intersect B_i, we have preserved the desirable local minima / local maxima preserving property (I.4).

Our algorithm allows the user to specify all three choices (raw, conservative, and weighted average) and in practice we have found weighted average is the most-used choice.

Our strategy is to compute the values V_{ij} exactly when both A_j and B_i are convex planar polyhedra using the robust algorithm described in Section I.3.4. If however A_j or B_i have some nonplanar faces (the chief case being when the faces are bilinear as part of a trilinear hexahedral element), the computation of the exact volume would be prohibitively expensive. Instead we 'planarize' any nonplanar faces (i.e., approximate the nonplanar facets with planar ones) and then compute the exact intersection volumes $V_{ij} \equiv |B_i' \cap A_j'|$, where B_i' and A_j' are the 'planarized' versions of B_i and A_j.

I.3 Algorithms

An efficient algorithm to map cell-based functions between unstructured meshes requires efficient evaluation of the approximate volumes in (I.5). This task is naturally broken up into two parts:

Find intersections: Find out for which i, j we have $V_{ij} \neq 0$. (Here we assume the approximations V_{ij} will only be nonzero if the exact volumes $|B_i \cap A_j|$ are nonzero.) V_{ij} is a sparse matrix: it has far fewer than $n_B \cdot n_A$ nonzero entries. Looping over all $n_B \cdot n_A$ entries would be fatally inefficient and is unnecessary. Our approach has complexity $\mathcal{O}(N \log N)$ where N is the number of nonzero V_{ij}.

Compute Intersections: For nonzero $\widetilde{V_{ij}}$, compute the intersection volumes exactly if the elements are planar and approximately if they are nonplanar.

I.3.1 Finding Intersections

The idea of our algorithm is to use the assumption of face-connectedness of both unstructured meshes. This allows us to traverse the A and B meshes simultaneously, finding the nonzero V_{ij} values along the way. We first start by putting A_1 (i.e. element number one from mesh A) onto a stack. Now we proceed by popping off the stack the element on top of the stack (call it A_j) and we say that we have "visited" this element; we then place back on the stack all unvisited face-neighbors of A_j. We deal with A_j by finding all B_i such that $B_i \cap A_j \neq \emptyset$ and computing the corresponding approximate intersection volumes V_{ij}. Initially, for A_1, we start with B_1 and walk along the B mesh using the B mesh connectivity until we find a B_i such that $A_1 \cap B_i \neq \emptyset$. Then we do some more walking in the B mesh in the neighborhood of B_i until we have discovered all the B_i such that $A_1 \cap B_i \neq \emptyset$. We continue by popping another element A_j off the stack. We then walk towards this new element on the B mesh starting not from B_1 but from the last element B_i that had a nonzero intersection with the previously considered element from the A mesh.

We thus are effectively performing a coordinated walk on the A and B meshes, and this should have a complexity $\mathcal{O}(N)$, where N is the number of nonzero V_{ij}. After all the elements of mesh A have been visited, we reorder the volume contributions V_{ij} so that they can be put in row-packed sparse matrix form. This reordering process employs a heapsort [92] and takes $\mathcal{O}(N \log N)$ operations.

Algorithm 1 gives the 'outer' algorithm where a 'walk' is performed on the face-connected A mesh.

The subroutine `walk_mesh_pt` (Algorithm 2) accomplishes the task of "walking" from a starting element B along a path of face-connected elements and ending at a new element B that contains the point $\mathbf{x}$.

The idea of this algorithm is that given a current element B, the point is tested against all the faces of B. (Note: in the case of nonplanar bilinear faces, we actually test $\mathbf{x}$ against the plane that passes through the midpoints of the 4 linear edges bounding the face. It is easily shown that the 4 midpoints are coplanar.) If $\mathbf{x}$ is on the wrong side of face k, then B doesn't contain $\mathbf{x}$ and we must move to a new element which becomes the "new" B. We usually move towards $\mathbf{x}$ by choosing the neighboring element across face k. Since nonconvex domains may necessitate somewhat circuitous walking paths, we prove the following lemma.

Algorithm 1: Compute_IntVols

`Compute_IntVols(Mesh_A,Mesh_B,IntVols)`
[Output `IntVols` is a sparse matrix representation of the
approximate overlap volumes $V_{ij} \approx \mathrm{volume}(B_i \cap A_j)$ for elements
B_i in `Mesh_B` and A_j in `Mesh_A`]

`vol_list`$\leftarrow \emptyset$
[`vol_list` will be a list of 3-tuples (i, j, V_{ij})
which will grow as intersection volumes are computed]
`OnStackA`$(A_j) \leftarrow$`.false.` for all elements A_j in `Mesh_A`
Stack $S_A \leftarrow \emptyset$
`SeedEltB`$\leftarrow B_1$ [first element of `Mesh_B`]
Put Element A_1 [first element of `Mesh_A`] on stack S_A
`OnStackA`$(A_1) \leftarrow$ `.true.`
Do while stack S_A nonempty
    Pop Element A_j off stack S_A
    Loop over the faces k of Element A_j
        If there is an element E_{opp} sharing face k with A_j then
        If (`.not.OnStackA`(E_{opp})) then
            Put E_{opp} on stack S_A
            `OnStackA`$(E_{\mathrm{opp}}) \leftarrow$ `.true.`
    $\mathbf{x}_A \leftarrow$ centroid of vertices of Element A_j
    Call `walk_mesh_pt`(`SeedEltB`, `Mesh_B`, $\mathbf{x}_A$)
    [Walk on `Mesh_B`, updating `SeedEltB`, until $\mathbf{x}_A \in$ `SeedEltB`]
    Call `get_vols_around_elt`(A_j, `SeedEltB`, `Mesh_B`, `vol_list`)
    [Append to `vol_list` 3-tuples (i, j, V_{ij}) where $V_{ij} \approx \mathrm{volume}(B_i \cap A_j)$ for
    $B_i =$`SeedEltB` and other B_i in the neighborhood of `SeedEltB` that intersect A_j]
Sort `vol_list` into sparse row-packed matrix `IntVols`
[Use heapsort to sort (i, j, V_{ij}) so that i's are in increasing order (and
if equal i, then j's are in increasing order).]
Return

Algorithm 2: Walk_mesh_pt [Walking on `Mesh` to point $\mathbf{x}$]

`Walk_mesh_pt` (E,`Mesh`,$\mathbf{x}$)

Initially place E on stack S
Do while S nonempty
    Pop E off stack S
    `viable` $\leftarrow$ `.true.`
    $E_{\rm top} \leftarrow 0$
    Loop over the faces k of Element E
        If (`viable`) then
            If there is an element $E_{\rm opp}$ sharing face k with E then
                If $\mathbf{x}$ on "wrong" side of face k (so that $\mathbf{x} \notin E$) then
                    `viable` $\leftarrow$ `.false.`
                    If $E_{\rm opp}$ was never put on stack S then
                        $E_{\rm top} \leftarrow E_{\rm opp}$ [$E_{\rm top}$ will be neighbor of E put on S *last* so it will be the *next* element considered]
                Else
                    If $E_{\rm opp}$ was never put on S then
                        Put $E_{\rm opp}$ on S
            Else
                If $\mathbf{x}$ on wrong side of face i then
                    `viable` $\leftarrow$ `.false`
        Else
            If there is an element $E_{\rm opp}$ sharing face k with E then
                If $E_{\rm opp}$ was never put on S then
                    Put $E_{\rm opp}$ on S
    If `viable` then
        Return [success: $\mathbf{x} \in E$]
    Else
        If $E_{\rm top} \neq 0$ then
            Put $E_{\rm top}$ on S
$E \leftarrow 0$ [failure]
Return

Algorithm 3: Get_vols_around_elt [Walking on `Mesh_B` to find intersection volumes with element A_j from `Mesh_A`]

`Get_vols_around_elt`(A_j,`SeedEltB`,`Mesh_B`,`vol_list`)

$\mathcal{P} \leftarrow$ `planarization`(A_j)
[Replace curved faces of A_j with interpolating planes]
Initially place `SeedEltB` on stack S_B
Do while stack S_B nonempty
    Pop element B_i off stack S_B
    Loop over `nfac` faces k of B_i
        Approximate face k with plane P_k
    [Now $B_i \approx \bigcap_k^{\texttt{nfac}} D_k$ where D_k is the halfspace bounded by P_k]
    `PlaneCuts`$(k) \leftarrow$`.false.`, $\quad 1 \le k \le$ `nfac`
    `emptyintersection` $\leftarrow$.`false`.
    $\mathcal{P}_{\rm int} \leftarrow \mathcal{P}$
    Do k = 1, `nfac`
        If $D_k \cap A_j = A_j$ then
            cycle [immediately start next iteration of 'do loop']
        Else if $D_k \cap A_j = \emptyset$ then
            `emptyintersection` $\leftarrow$.`true`.
            exit [exit 'do loop']
        Else
            `PlaneCuts`$(k) \leftarrow$`.true.`
            Call `Plane_Poly_Int3D`$(P_k, \mathcal{P}_{\rm int}, \mathcal{P}_{\rm tmp})$
            [Computes intersection polyhedron $\mathcal{P}_{\rm tmp} = \mathcal{P}_{\rm int} \cap D_k$]
            $\mathcal{P}_{\rm int} \leftarrow \mathcal{P}_{\rm tmp}$
            If $\mathcal{P}_{\rm int} = \emptyset$ then
                `emptyintersection` $\leftarrow$.`true`.
                exit [exit 'do loop']
    If `emptyintersection` $=$.`false`. then
        `NewVol`$\leftarrow$ `Volm_Poly3D`$(\mathcal{P}_{\rm int})$ [Volume of $\mathcal{P}_{\rm int}$]
        Append 3-tuple $(i, j,$ `NewVol`$)$ to `Vol_list`
        Do $k = 1,$ `nfac`
            If (`PlaneCuts(`k`)`) then
                If there is an element sharing face k with B_i then
                    $E_{\rm opp} \leftarrow$ element opposite from face k of B_i
                    If $E_{\rm opp}$ never before placed on stack S_B then
                        Place $E_{\rm opp}$ on stack S_B
Return

Algorithm 4: Map_cell_field [Map field f^A to field f^B]

`Map_cell_field` (f^A, f^B, `IntVols`, `exactly_conservative`, `preserve_constants`)

[The sparse matrix `IntVols` is assumed stored as follows:
There are 3 arrays: `rowoff, col,` and `vol`.
Intersection volumes pertaining to B_i (row i) are in
array subset `vol(rowoff(i):rowoff(i+1)-1)`
For k'th element of `vol`, we have that `col(k)` gives the
column number $j =$ `col(k)`, pertaining to A_j, so that
`vol(k)`$= V_{ij} \approx \text{volume}(B_i \cap A_j)$.]

[We assume the following quantities are available:
$|A_j|$ = volume(A_j) $1 \le j \le n_A$
$|B_i|$ = volume(B_i) $1 \le i \le n_B$
$\texttt{RowSum}(i) = \sum_{j=1}^{n_A} V_{ij} \quad 1 \le i \le n_B$
$\texttt{ColSum}(j) = \sum_{i=1}^{n_B} V_{ij} \quad 1 \le j \le n_A$]

[We assume `exactly_conservative` and `preserve_constants` are not both `.true.`]

Do $i = 1, n_B$
  $f_i^B \leftarrow 0$
  Do $k = \texttt{rowoff}(i), \texttt{rowoff}(i+1) - 1$
    $j \leftarrow \texttt{col}(k)$
    If (`exactly_conservative`) then
      $f_i^B \leftarrow f_i^B + \texttt{vol}(k) \times \frac{|A_j|}{\texttt{ColSum}(j)} \times f_j^A$
    Else
      $f_i^B \leftarrow f_i^B + \texttt{vol}(k) \times f_j^A$
  If (`preserve_constants`) then
    $f_i^B \leftarrow f_i^B / \texttt{RowSum}(i)$
  Else
    $f_i^B \leftarrow f_i^B / |B_i|$
Return

Lemma 1. In Algorithm 2, if $\mathbf{x}$ is in some element, we are given a valid starting element, and there is an element containing $\mathbf{x}$, Algorithm 2 will successfully find the element containing $\mathbf{x}$.
Proof: Since we assume the mesh is face-connected, there is a face-connected path

$$E = E_0, E_1, E_2, \ldots, E_n \ni \mathbf{x} \tag{I.9}$$

from the starting element E to E_n which should be the new E returned by the algorithm. Algorithm 2 uses a stack S and initially E_0 is placed on the stack. Every time an element E is popped off the stack, it is checked for the property $\mathbf{x} \in E$, and if true, we are done. Otherwise, all the neighbors of E that have never been on the stack before are put on the stack. Since an element can only be placed at most once on the stack, the only way the algorithm can fail is that eventually the stack is empty and E_n has not been found. In this case, there is a finite set of elements that were visited before the algorithm failed. This set is

$$\mathcal{E}_{\text{visited}} = \{E \mid E \text{ was popped off } S\}.$$

Let E_i be the element in the sequence in (I.9) that belongs to $\mathcal{E}_{\text{visited}}$ and has maximal index in the sequence in (I.9). When E_i was popped off the stack, E_{i+1} was either placed on the stack because E_i and E_{i+1} are face-neighbors or it wasn't because it had previously been placed on the stack. Either way, this contradicts the assumption that $E_i \in \mathcal{E}_{\text{visited}}$ has maximal index. Q.E.D.

It is similarly proven that in Algorithm 1, the traversal of Ω_A using the stack S_A successfully visits all elements in mesh A once and terminates.

We call Algorithm 3 with `SeedEltB` equal to the element B_i that was found to contain element A_j by Algorithm 2. For the case that A_j and all B_i are planar and convex, we prove the following lemma and defer discussion of the nonplanar case to section I.3.3.

Lemma 2. In Algorithm 3, for the case that A_j and all B_i are planar, convex polyhedra, and assuming that $A_j \cap$ `SeedEltB` $\neq \emptyset$, we have that all intersections of A_j with Mesh B will be found and deposited into `vol_list`.
Proof: Since we assume A_j is planar and convex, the `planarization` operation does nothing, so that $\mathcal{P} = A_j$. `SeedEltB` is initially placed on stack S_B. The algorithm works by popping an element B_i from Mesh B off the stack, computing the intersection volume $V_{ij} = |B_i \cap A_j|$ and then placing this (i, j, V_{ij}) information into `Vol_list` if $V_{ij} > 0$. The element B_i is stored as the intersection of `nfac` half-spaces. That is, $B_i = \cap_{k=1}^{\texttt{nfac}} D_k$, where D_k is the half-space bounded by P_k which is the plane containing the k'th boundary face. The intersection is performed using routine `Plane_Poly_Int3D` which intersects a half-space with a planar convex polyhedron. B_i is assigned to $\mathcal{P}_{int}$, the "intersection polyhedron", and this polyhedron is then intersected in order with $D_1, D_2, \ldots, D_{\texttt{nfac}}$, each intersection resulting in a potentially smaller $\mathcal{P}_{int}$. At the end of this process, if $\mathcal{P}_{int} \neq \emptyset$, routine `Volm_Poly3D` computes the volume of $\mathcal{P}_{int} = B_i \cap A_j$ by decomposition into tetrahedra. If $V_{ij} > 0$, then for each face k of B_i, we place the adjacent element E_{opp} on S_B if and

only if (i) the element E_{opp} actually exists and has never been placed on the stack before, and (ii) the plane P_k containing the common boundary facet k between B_i and E_{opp} nontrivially divides A_j. (Condition (ii) is checked by seeing if all the nodes of A_j lie on either side of P_k.) Clearly, if P_k does not intersect A_j, then it is impossible for both $B_i \cap A_j \neq \emptyset$ and $E_{opp} \cap A_j \neq \emptyset$. Since $B_i \cap A_j \neq \emptyset$, we have that $E_{opp} \cap A_j = \emptyset$, and we are justified in not putting E_{opp} onto the stack. Proceeding in this fashion, it is clear that this algorithm will find all intersections $V_{ij} = |B_i \cap A_j|$ if the set

$$\mathcal{B}_{A_j} = \{B_i \mid |B_i \cap A_j| > 0\}$$

is face-connected. Now if B_{i_1} and B_{i_2} are in $\mathcal{B}_{A_j}$, let $\mathbf{x}_1 \in \text{interior}(B_{i_1} \cap A_j)$ and $\mathbf{x}_2 \in \text{interior}(B_{i_2} \cap A_j)$. Then since A_j is connected, we can connect $\mathbf{x}_1$ to $\mathbf{x}_2$ by a curve Γ lying entirely within $\text{interior}(A_j)$. Since $A_j \subseteq \cup_{i=1}^{n_B} B_i$, Γ traces a path from $\mathbf{x}_1$ to $\mathbf{x}_2$ entirely within Mesh B as well. If necessary, we can perturb Γ so that it doesn't intersect any of the vertices or edges in Mesh B and "cleanly" intersects the facets in Mesh B (i.e., so that the curve does not run tangentially within any facet for any positive distance). Since the original Γ lies entirely within the interior of A_j, and the necessary perturbations (to avoid vertex and edge intersections and to assure clean face intersections) can be arbitrarily small, the perturbed Γ can be assumed to lie entirely within the interior of A_j as well. In this case, all the B_i encountered by Γ when travelling from $\mathbf{x}_{i_1}$ to $\mathbf{x}_{i_2}$ form a face-connected sequence and all are members of $\mathcal{B}_{A_j}$. This shows that $\mathcal{B}_{A_j}$ is face-connected. Q.E.D.

Algorithm 4 is the "payoff" algorithm where we take our intersection volumes that have been previously computed in Algorithms 1-3 and use them to map a cell field f^A on mesh A to a cell field f^B on mesh B. At the end of Algorithm 1, the overlap volumes (i, j, V_{ij}) stored in `vol_list` were sorted into a row-packed sparse matrix `IntVols` which is actually three arrays `rowoff`, `col`, and `vol` which represent a standard row-packed sparse matrix data structure. We also assume that the volumes of the elements in both meshes and the column and row sums of the V_{ij} have been computed and are available. Algorithm 4 takes the input cell field f^A, along with `IntVols`, and the two logical flags `exactly_conservative`, and `preserve_constants` and then maps f^A to f^B by sparse matrix multiplication. If neither boolean flag is true, the "raw" mapping (I.6) is used. If `exactly_conservative` is `.true.` then we use the exactly conservative mapping (I.7). If `preserve_constants` is `.true.`, we use the weighted average mapping (I.8). (We call the flag `preserve_constants` because (I.8) preserves constants in the sense that if $f^A \equiv k$, then $f^B \equiv k$.)

It is not legal for both `exactly_conservative` and `preserve_constants` to be `.true.` Indeed, in order to obtain both of those properties simultaneously, an iteration would be necessary and in fact both those properties are inconsistent with each other unless $\text{Volume}(\Omega_A) = \text{Volume}(\Omega_B)$ exactly.

I.3.2 Practical Geometry Considerations

I.3.2.1 Sloppiness at the Boundary

In practice we cannot have $\Omega_A = \Omega_B$ exactly. For example, if we alternately discretize some sphere S using tetrahedra on mesh A and hexahedra on mesh B, we will not usually have $\Omega_A = \Omega_B$ and certainly not $\Omega_A = S$ or $\Omega_B = S$. This is assuming the tetrahedra have planar facets and the hexahedra have bilinear facets. If the boundary nodes of the hex and tet grids reside on ∂S, then $\mathrm{vol}(S) > \mathrm{vol}(\Omega_A)$ and $\mathrm{vol}(S) > \mathrm{vol}(\Omega_B)$. In fact, even with a large number of elements, although

$$\mathrm{vol}(S) \approx \mathrm{vol}(\Omega_A) \approx \mathrm{vol}(\Omega_B),$$

probably none of these volumes will be exactly equal. In this scenario, parts of the tet mesh might slightly poke out of Ω_B. When Algorithm 1 calls Algorithm 2, it is called with the centroids $\mathbf{x}$ of some elements in Ω_A. (Here the "centroid" is the cheap-to-compute average position of the vertices of the element.) Since the centroid of A_j is at least a distance $\mathrm{ir}(A_j)$ from points on the surface of A_j (where $\mathrm{ir}(A_j)$ denotes the radius of the largest sphere inscribed in A_j centered on the centroid), we have that the element A_j can stick out of Ω_B by at least $\mathrm{ir}(A_j)$ and the algorithm will still work. So our formal assumption $\Omega_A \subseteq \Omega_B$ is relaxed so that elements of mesh A can stick out of Ω_B to this small degree.

I.3.2.2 Element Blocks

The unstructured mesh data structure supports the concept of "element blocks" which are effectively a decomposition of the mesh into regions. Thus, instead of there being a single Ω_A, there is a decomposition

$$\Omega_A = \bigcup_{k=1}^{K_A} \Omega_A^{e_k}$$

and a decomposition

$$\Omega_B = \bigcup_{k=1}^{K_B} \Omega_B^{f_k}.$$

The simplest case is $e_k = k$ and $f_k = k$, but the element block numbers need not be sequential. In the general case, we assume that $\{e_k \mid 1 \le k \le K_A\} \subseteq \{f_k \mid 1 \le k \le K_B\}$ and there is for each $1 \le k \le K_A$ a correspondence between $\Omega_A^{e_k}$ and $\Omega_B^{e_k}$. We assume that $\Omega_A^{e_k} \subseteq \Omega_B^{e_k}$, or at least to within a certain tolerance as explained in the previous discussion. Now the actual implentation of Algorithm 4 has an input parameter `strict` which determines if the algorithm will pay attention to element blocks. This parameter is by default `.true.`. In this case, if we also have `exactly_conservative=.true.`, our algorithm assures conservation within each region,

$$\int_{\Omega_B^{e_k}} f^B \, dV = \int_{\Omega_A^{e_k}} f^A \, dV.$$

If `strict=.true.` and `preserve_constants=.true.`, our mapping will assure preservation of independent constants in each region:

$$f^A\big|_{\Omega_A^{e_k}} \equiv \text{const}_k \Rightarrow f^B\big|_{\Omega_B^{e_k}} \equiv \text{const}_k.$$

Equivalently, the mapped field value f_i^B will be a weighted average of the source field values f_j^A that come from elements A_j that intersect B_i and whose element block numbers match that of B_i.

I.3.2.3 Relaxation of Face-Connected Mesh Assumption

In practice, element blocks can be disconnected from each other. We have enhanced the robustness of our actual implementations of Algorithm 1 so that Mesh A need not be all face-connected. Also, the actual implementation of Algorithm 2 has also been enhanced so that Mesh B need not be face-connected. In the actual implementations of these algorithms, if a connected component of the mesh has been searched without success, we search other connected components, until we have searched all connected components. Clearly, this will endanger our algorithm's $\mathcal{O}N \log N$ time complexity bound. However, if we have for each element block inclusion $\Omega_A^{e_k} \subseteq \Omega_B^{e_k}$ that both element blocks $\Omega_A^{e_k}$ and $\Omega_B^{e_k}$ are face-connected, then we will traverse the element blocks of mesh A efficiently with Algorithm 1 and we will possibly have to search the whole Mesh B in Algorithm 2 whenever the point $\mathbf{x}$ provided as input to the algorithm has switched to a new face-connected component of Mesh A. The additional time spent is thus bounded by $(K_A - 1)n_B$, where K_A is the number of face-connected element blocks of Mesh A. If we call K_A a "constant", then since $n_B < N$, we have the additional time complexity is $\mathcal{O}(N)$, and so the overall time complexity bound of $\mathcal{O}(N \log N)$ still stands for grid mapping. Algorithm 3 still assumes face-connectedness of the set of nonzero intersection volumes

$$\mathcal{B}_{A_j} = \{B_i \mid |B_i \cap A_j| > 0\}.$$

This is not a problem if for each element block inclusion $\Omega_A^{e_k} \subseteq \Omega_B^{e_k}$ we have that $\Omega_B^{e_k}$ is face-connected; otherwise, intersection volumes may be missed by Algorithm 3.

So in summary, we have relaxed the face-connectedness requirements of Mesh A and Mesh B, but to preserve $\mathcal{O}(N \log N)$ complexity and to preserve the accuracy of the intersection volume calculation, we still require that for each element block inclusion $\Omega_A^{e_k} \subseteq \Omega_B^{e_k}$ that both element blocks $\Omega_A^{e_k}$ and $\Omega_B^{e_k}$ are face-connected.

I.3.2.4 Gap Elements

Certain applications in TRUCHAS require *gap elements* that have zero volume but are face-connected to the rest of the mesh. Thus, for example, a face-connected path between two normal, non-gap elements may

involve traversal through a zero-thickness gap element. In the grid mapping software, we say that element E is a gap element if

$$\text{volume}(E) \leq \epsilon_V \text{ volume}(\text{ball}(E)),$$

where ball(E) is the smallest ball centered at the centroid of the vertices of E which contains all the vertices of E, and ϵ_V is a small tolerance. (In the software, this tolerance is called `eps_vol_frac` and is currently set to 10^{-8}.)

Algorithms 1-3 will never deal with gap elements, but instead will always "step around" a gap element and look at its neighbors. As a consequence, there are no entries corresponding to gap elements in the volume intersection matrix V_{ij}.

I.3.3 Treatment of nonplanar faces

The polyhedral zoo of elements processed by our algorithm all have facets which are either planar triangles or bilinear quadradrilatals. The elements possessing some or all bilinear quads for facets are hexahedra, pyramids, and triangular prisms. The key subroutine `Plane_Poly_Int3D` described in Section I.3.4 performs intersections of half-spaces (defined by oriented planes) with planar polyhedra. Thus we need to "planarize" any nonplanar facets before calling `Plane_Poly_Int3D`.

We proceed as follows. First it is easy to prove that for an element E and a bilinear quad facet k, there exists a "best fit" plane P_k that passes through the midpoints of all four edges of the facet as well as through the centroid of the four vertices. In fact, if the vertices of the quad are labelled $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4$ in cyclical order, the normal of this plane is

$$\hat{\mathbf{n}}_k = \pm \frac{(\mathbf{x}_2 - \mathbf{x}_4) \times (\mathbf{x}_3 - \mathbf{x}_1)}{||(\mathbf{x}_2 - \mathbf{x}_4) \times (\mathbf{x}_3 - \mathbf{x}_1)||}. \tag{I.10}$$

(We choose the sign so that $\hat{\mathbf{n}}_k$ is an "inward" normal pointing towards the interior of the element E.) The equation of the plane P_k is thus

$$\hat{\mathbf{n}}_k \cdot (\mathbf{x} - \mathbf{x}_k^{\text{cen}}) = 0,$$

where $\mathbf{x}_k^{\text{cen}}$ is the centroid of the vertices of the face, $\mathbf{x}_k^{\text{cen}} = (\mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3 + \mathbf{x}_4)/4$. Now let D_k be the half-space bounded by P_k with normal $\hat{\mathbf{n}}_k$ pointing towards the interior of D_k. (If instead k is a planar triangular face, then D_k is simply the half-space bounded by the (correctly oriented) plane P_k containing face k.) We say the *planarization* of E is given by

$$planarization(E) = \cap_{k=1}^{\texttt{nfac}} D_k, \tag{I.11}$$

where `nfac` is the number of faces of E. $planarization(E)$ is our stand-in for E when intersection volumes need to be computed in Algorithm 3. The element A_j is initially planarized, and then it is intersected with

the "best fit" planes of the faces of element B_i in order to compute the approximation $V_{ij} \approx B_i \cap A_j$. If E is planar, convex, then $planarization(E) = E$ and no harm is done. If E has slightly nonplanar facets and the approximating planes do not intersect with dihedral angles greater than π, then $planarization(E)$ will be a reasonably good approximation to E. If E has highly nonplanar facets or if the approximating planes P_k have some dihedral angle intersections that exceed π, then $planarization(E)$ will be a poor approximation to E so that the approximation V_{ij} may not be close to the true intersection volume, but this is a situation that should not occur for many elements in well-constructed meshes.

Now regarding our treatment of Algorithm 2, suppose two elements B_{i_1} and B_{i_2} share the same quad face. Say that face is k_1 in an ordering of the faces of E_{i_1} and say that face is k_2 in an ordering of the faces of E_{i_2}. Then we have that P_{k_1}=P_{k_2} (the planarizations of the two adjacent elements agree on their common face), $\hat{\mathbf{n}}_{k_1} = -\hat{\mathbf{n}}_{k_2}$, and $\{D_{k_1}, D_{k_2}\}$ form a nonoverlapping partition of the space $\mathbb{R}^3$. However, it does *not* follow from this that

$$\{planarization(B_1), planarization(B_2), \ldots, planarization(B_{n_B})\} \tag{I.12}$$

would form a nonoverlapping polyhedral decomposition of the domain $\Omega_B = \cup_i B_i$. In fact, there can be "holes" in the way (I.12) covers Ω_B in the neighborhood of edges. (See Figure I.1.) This means that in Algorithm 2, if we interpret the statement

> If $\mathbf{x}$ on "wrong" side of face $k \ldots$

to mean

> If $\mathbf{x}$ on "wrong" side of plane $P_k \ldots$, (I.13)

then we have a rapid test for determining whether to traverse across a face, but it is conceivable that $\mathbf{x}$ in mesh A will not be locatable in mesh B, if it falls in one of the "holes", such as depicted in Figure I.1. To prevent falling into holes, for the purposes of Algorithm 2, we use (I.13), but with respect to an element B and a face k, we move the position of P_k back so that all of face k is on the "correct" side of P_k. With this readjustment, the set (I.12) will form a cover of Ω_B with some overlaps. This means that in the neighborhood of a face, point $\mathbf{x}$ may be considered to be inside of more than one element. This is acceptable, since the first such element encountered will be returned by Algorithm 2 and will be used as the seed element for Algorithm 3.

I.3.4 Computing Intersections

We employ an algorithm for computing the intersection volume $V_{ij} = |B_i \cap A_j|$ exactly in the case that both elements A_j from mesh A and B_i from mesh B are convex planar polyhedra. Two key routines are used: `Plane_Poly_Int3D` intersects a half-space with a planar convex polyhedron and returns the intersection which, if nonempty, is itself a planar convex polyhedron. `Volm_Poly3D` computes the volume of a planar convex polyhedron by decomposing it into tetrahedra and then summing the volumes of the tetrahedra.

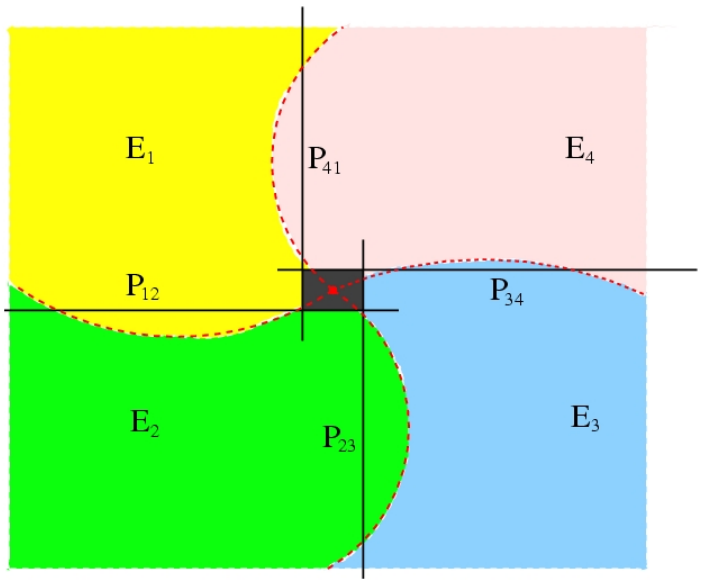


Figure I.1: Hole created by planarization: Curved interface between elements E_1, E_2 is replaced by common "best fit" plane P_{12} and similarly for the other interfaces. This creates a hole—a region shown by the gray area that would not be considered to be within any element. (2-D schematic of the 3-D situation.)

I.3.5 Weighted Average vs. Exactly Conservative

Since the "weighted average" mapping ((I.8) invoked with `preserve_constants` in Algorithm 4) has the desirable property of not creating any new local minima or maxima (I.4), it is the mapping we have used most often in practice. A typical case for our Truchas code is that we have a set of "volume fraction" fields $f^{(1)}, f^{(2)}, \ldots, f^{(P)}$ that have the property

$$\sum_{p=1}^{P} f_i^{(p)} = 1, \quad \forall i. \tag{I.14}$$

Since "the sum of the weighted average" is the "weighted average of the sum", it is easy to prove that the mapped fields (when using the weighted average mapping (I.8)) will obey the "summing to 1" property (I.14) as well. This property will not be preserved by the "raw" or "exactly conservative" mappings.

Since it is tempting to use the non-conservative weighted average mapping in most situations, we compute a bound on the L_1 error between the conservative and weighted average mappings. Let $f(\mathbf{x})$ denote the source cell field on mesh A; we take this to be a density of some sort of substance and so make the assumption that $f(\mathbf{x}) \geq 0$. Let f^{cons} and f^{wa} be the conservative and weighted average fields derived over mesh B by using

(I.7) and (I.8) respectively. Then

$$\begin{aligned}
&\int_{\Omega_B} |f^{\text{cons}}(\mathbf{x}) - f^{\text{wa}}(\mathbf{x})|\, dV \\
&\quad = \sum_i |B_i| \left| f_i^{\text{cons}} - f_i^{\text{wa}} \right| \\
&\quad = \sum_i |B_i| \left| \frac{1}{|B_i|} \sum_j V_{ij} \frac{|A_j|}{\sum_m V_{mj}} f_j - \frac{1}{\sum_k V_{ik}} \sum_j V_{ij} f_j \right| \\
&\quad = \sum_i \sum_j \left| V_{ij} \left(\frac{|A_j|}{\sum_m V_{mj}} - \frac{|B_i|}{\sum_k V_{ik}} \right) f_j \right| \\
&\quad \leq \sum_i \sum_j V_{ij} f_j \left(\left| \frac{|A_j|}{\sum_m V_{mj}} - 1 \right| + \left| \frac{|B_i|}{\sum_k V_{ik}} - 1 \right| \right) \\
&\quad = \sum_j \left(f_j \sum_i V_{ij} \right) \left| \frac{|A_j|}{\sum_m V_{mj}} - 1 \right| + \sum_i \left(\sum_j V_{ij} f_j \right) \left| \frac{|B_i|}{\sum_k V_{ik}} - 1 \right| \\
&\quad \approx \sum_j \left(\int_{A_j} f(\mathbf{x})\, dV \right) \left| \frac{|A_j|}{\sum_m V_{mj}} - 1 \right| + \sum_i \left(\int_{B_i} f(\mathbf{x})\, dV \right) \left| \frac{|B_i|}{\sum_k V_{ik}} - 1 \right| \\
&\quad \equiv \mathbf{F}^A \cdot \mathbf{E}^A + \mathbf{F}^B \cdot \mathbf{E}^B
\end{aligned} \tag{I.15}$$

Here, we have defined

$$\mathbf{F}_j^A \equiv \int_{A_j} f(\mathbf{x})\, dV \qquad \mathbf{F}_i^B \equiv \int_{B_i} f(\mathbf{x})\, dV$$

as being, respectively, the integral of f over an element of Mesh A, and the integral of f over an element of Mesh B. We have defined

$$\mathbf{E}_j^A \equiv \left| \frac{|A_j|}{\sum_m V_{mj}} - 1 \right| \qquad \mathbf{E}_i^A \equiv \left| \frac{|B_i|}{\sum_k V_{ik}} - 1 \right|$$

as being, the relative volume errors caused by our decomposition of elements A_j in mesh A and elements B_i in mesh B, respectively.

Now since

$$\begin{aligned}
\left| \int_{\Omega_A} f(\mathbf{x})\, dV - \int_{\Omega_B} f^{\text{wa}}(\mathbf{x})\, dV \right| &= \left| \int_{\Omega_B} f^{\text{cons}}(\mathbf{x})\, dV - \int_{\Omega_B} f^{\text{wa}}(\mathbf{x})\, dV \right| \\
&\leq \int_{\Omega_B} |f^{\text{cons}}(\mathbf{x}) - f^{\text{wa}}(\mathbf{x})|\, dV,
\end{aligned}$$

we have that by (I.15), $\mathbf{F}^A \cdot \mathbf{E}^A + \mathbf{F}^B \cdot \mathbf{E}^B$ is also bound for the conservation error of f^{wa}.

So we can use f^{wa} and be sufficiently conserving if the relative volume error vectors $\mathbf{E}^{\mathbf{A}}$ and $\mathbf{E}^{\mathbf{B}}$ are sufficiently small. In fact, with our algorithm, the error vector entries will be identically zero for planar-faceted elements that are completely covered by planar-faceted elements from the other mesh. Errors creep in the more the elements are curved, and the more the elements from the two meshes fail to match up at the boundary. The worst case would be mapping a field f which is only nonzero near a curved boundary $\partial\Omega_A$ and where the characteristic element sizes in Ω_A, Ω_B differ greatly. In this case, the error vectors $\mathbf{E}^A$, $\mathbf{E}^B$ would be large (due to poor matching of the meshes at $\partial\Omega_A$) precisely on those elements where the entries of the integrated quantity vectors $\mathbf{F}^A$, $\mathbf{F}^B$ are nonzero. In fact in our Truchas code we have a field representing the creation of inductive heating near the surface of a conductor being mapped from a tet mesh to a hex mesh. Only for this field do we feel the need to use the conservative map f^{cons} rather than the weighted average f^{wa}.

I.4 Numerical Results

We show results of our mapping algorithm on a "curved pipe with slit" geometry. In Figure I.2, we see a tet grid with 140651 elements on which a sample cell field $f((x, y, z)) = 1 + \sin(z)$ has been defined on the elements, where for each tet, the z value used to evaluate $f(z)$ for that tet is taken to be at the centroid of the tet. (In the figure, the geometry fits in the box $[0, 12] \times [0, 12] \times [-5, 5]$ and z is in the 'up' direction.) In Figure I.3, the field f has been mapped conservatively to a cell field over a hex mesh of the same geometry. The hex mesh has 42496 elements, and the number of intersection volumes V_{ij} that are nonzero in the map between the two meshes is 1032910. Here we have used the `exactly_conservative` option, so the integral of the field is same 513.9857790995 on both meshes using double precision computation. Although both tet and hex meshes attempt to discretize the same geometry, the boundary curvature causes the volumes of the two meshes to be different: 529.68 on the tet-discretized region and 529.34 on the hex discretized region. Thus, to accomodate the mesh volume differences and still be exactly conservative, it is not surprising the the maximum value of the field is 2.0252 on the hex mesh, which represents just over a 1% overshoot of the maximum value 2.0000 on the tet mesh. We note that the minimum value on the hex mesh is 0.0006 which does not undershoot the minimum value of 0.0000 on the tet mesh. This is because by (I.7), the values of the mapped field using the `exactly_conservative` option are still a positive linear combination of local source field values, so that negative values cannot arise when mapping a non-negative field.

In Figure I.4 we show the mapped field over the same hex mesh, where we have used the `preserve_constants` (weighted average) option. Here, the integral of the field over the destination mesh agrees only to 3 decimal places (it is 513.66); however, the global minima/maxima of the mapped field are 0.0006 and 1.9994 respectively, which are within the corresponding bounds of the source tet mesh.

The timings on an Apple G5 workstation with IBM XLF compiler are consistent with the claimed $N \log N$

Tet Elements n_A	Hex Elements n_B	Nonzero V_{ij} N	Time Alg. 1-3	Time Alg. 4		
				raw	cons	wted avg
2450	792	16042	0.36s	0.14ms	0.37ms	0.13ms
11761	6500	106127	2.73s	1.27ms	3.50ms	1.27ms
140651	42496	1032910	25.33s	23.98ms	91.60ms	24.13ms

Table I.1: Timings for 3 hexmesh-to-tetmesh mappings on curved pipe geometry

scaling for the building of the sparse mapping matrix (Algorithms 1-3). Using the same curved pipe geometry we computed mappings between three pairs of grids, always mapping from an all-tet mesh to an all-hex mesh. The timings are given in Table 1. We see that the time for computing the sparse matrix V_{ij} is in fact growing linearly with N, taking about 25μs per overlap entry. We note that Algorithms 1-3 need only be called *once*. Then any number of fields may be mapped by calling Algorithm 4 for each field. Algorithm 4 is very cheap compared to Algorithms 1-3 and it in fact has complexity $\mathcal{O}(N)$. Comparing the second and third rows in Table 1, it appears that Algorithm 4 time complexity is superlinear. However, this simply results from the fact that the vector of source field values over the tet mesh can completely fit in cache memory for the intermediate-sized case and cannot completely fit in cache memory for the largest-sized case on the Apple G5 architecture employed.

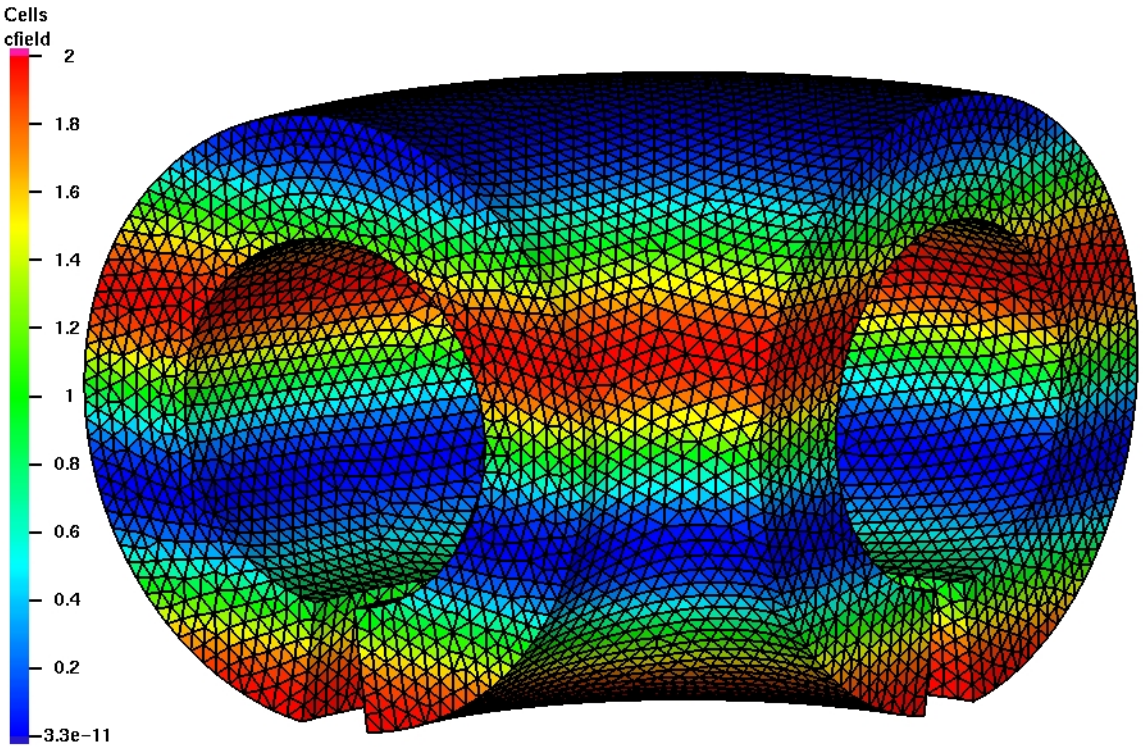


Figure I.2: Tet mesh on slitted curved pipe geometry showing source field $f(\mathbf{x}) = 1 + \sin(z)$

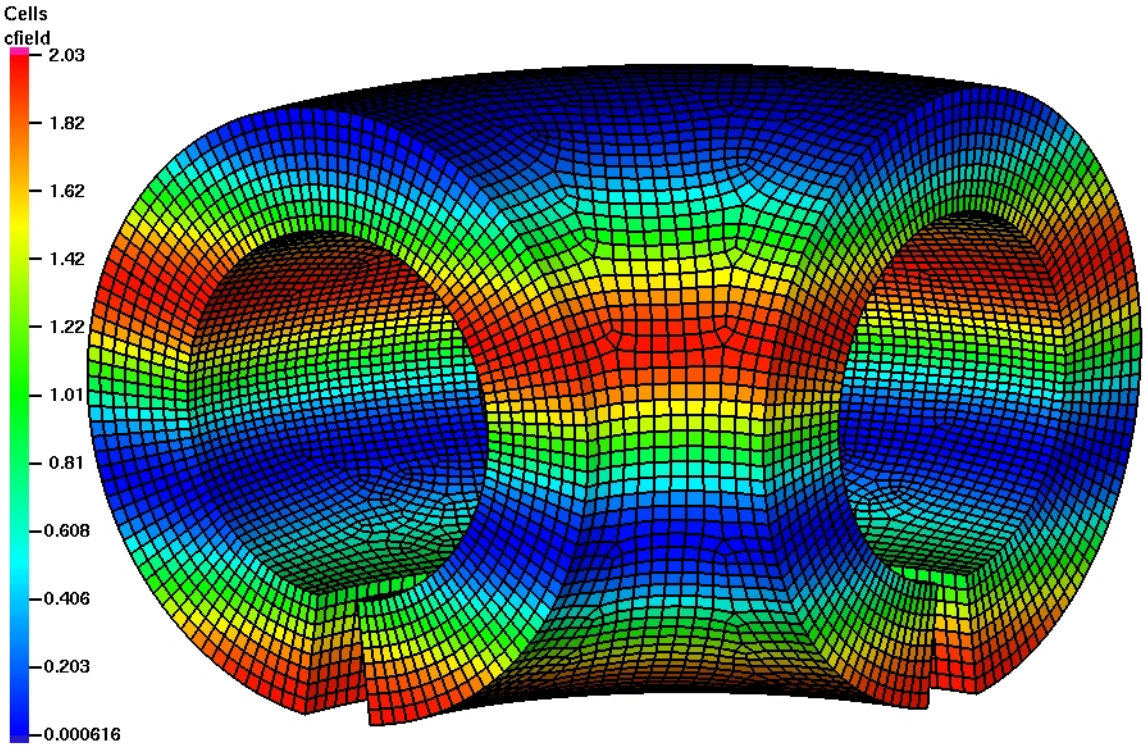


Figure I.3: Hex mesh on same geometry showing mapped field with `exactly_conservative` option.

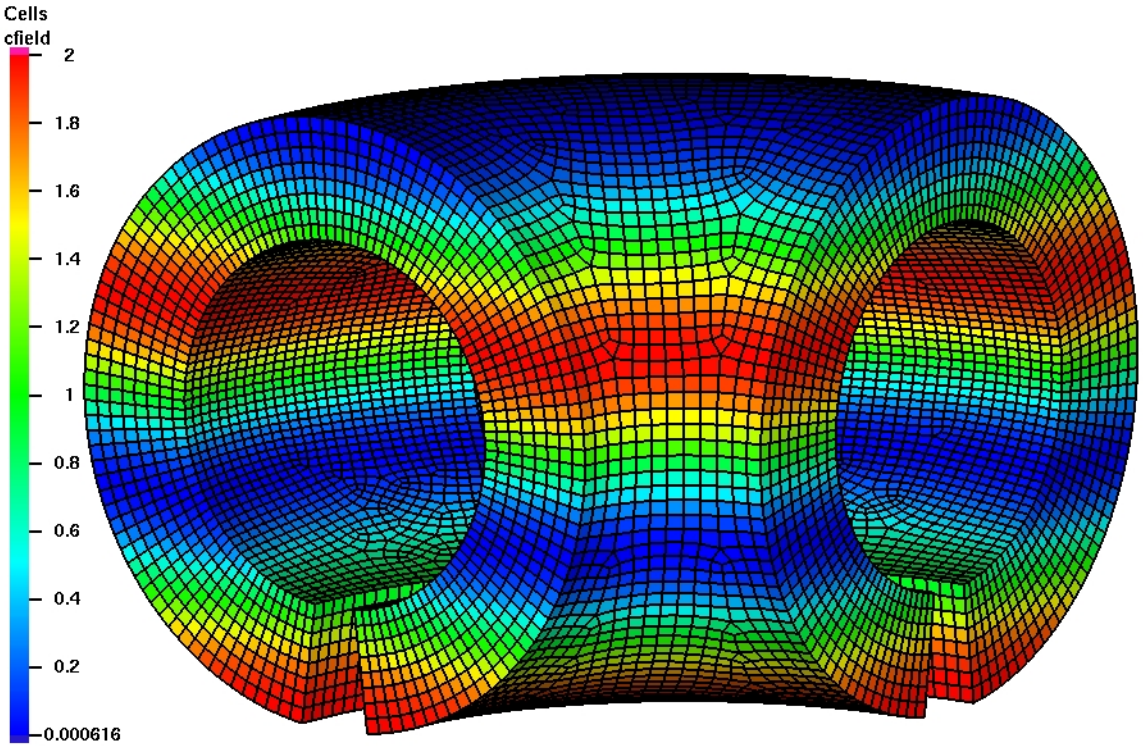


Figure I.4: Hex mesh on same geometry showing mapped field with `preserve_constants` (weighted average) option.

Appendix J

Nucleation and Growth

J.1 Rappaz-Thévoz Model with One-Way Coupling

We have implemented the method of Thévoz et. al. for modeling microstructure formation during solidification [93] using one-way coupling. In this method, computational cells that are actively undergoing solidification ($0 < f_s < 1$) are subjected to N explicit "microscopic" timesteps per implicit "macroscopic" timestep ($\delta t = \Delta t/N$). In this case, the macroscopic heat flow calculation provides the cell an enthalpy H and change in enthalpy ΔH for the cell, and the microscopic time evolution will evolve the nuclei number density n and average grain radius R in the cell. From these quantities, the solid fraction f_s can be updated. Since

$$\delta H = c_p \delta T - L \delta f_s, \tag{J.1}$$

we then can obtain the change in temperature δT for the micro-timestep. Performing this N times, we obtain the change in T for the macro-timestep. (Note: we avoid calling this change "ΔT", since the symbol ΔT is already reserved to denote the undercooling $T_l - T$ where T_l is the liquidus temperature.)

Thus as opposed to the usual assumed monotonic relationship between enthalphy and temperature, it is now possible to model *recalescence* where a sudden increase in f_s actually leads to an increase in temperature, even while heat is being removed from the cell. At this point, the release of latent heat due to solidification exceeds the heat being removed from the cell, the temperature rises, and nucleation of new grains is halted.

J.1.1 Nucleation Model

As in [93], it is assumed that nuclei number density at a given undercooling is given by the integral of a Gaussian nucleation site density distribution from zero undercooling to the current undercooling. (However,

nuclei number density increase is permanently halted at recalescence.) The Gaussian distribution is characterized by its standard deviation ΔT_σ, its mean (located at undercooling ΔT_N) and its integral $n_{\max}$. These three parameters are determined experimentally for each melt.

J.1.2 Growth Model

The solid fraction in the Rappaz-Thévoz model is given by

$$f_s(t) = n(t) \cdot \frac{4}{3}\pi R^3(t) \cdot f_i(t),$$

where f_i is the internal solid fraction which corresponds to the fraction of solid within the expanding spherical envelope of the grains (which have radius $R(t)$). We write

$$\delta f_s = n(t) \cdot \left(4\pi R^2 \delta R \cdot f_i + \frac{4}{3}\pi R^3 \cdot \delta f_i \right), \tag{J.2}$$

if one neglects the change δn. (One can see the absurdity of including a δn term as given by the product rule for derivatives, since such a term would imply the instantaneous creation of grains *at radius* R instead of radius 0. However, omission of the δn term does not change the fact that the model is still a simplification because it assumes a *single* radius R describes the distribution of grains in the nucleation and growth phases.)

Assuming (J.2), to compute δf_s, we need to know δR and δf_i. First, δR is naturally obtained by knowing the velocity v of the dendrite tip. This velocity is obtained from the model of Lipton et. al. [94] which relates v to the known undercooling of the grain. Second, the solute diffusion model of Rappaz and Thévoz [95] relates f_i to the the Peclet number $\mathrm{Pe} \equiv \frac{Rv}{2D}$ of the grain and the supersaturation at the dendrite tip. The supersaturation is

$$\Omega \equiv \frac{c^* - c_0}{c^*(1-k)},$$

which can be related to the undercooling by the phase diagram. (Here c^* is the concentration at the dendrite tips, c_0 is the alloy concentration, and k is the partition coefficient.) Since we know R, v, and D (solute diffusion coefficient), we can compute Pe. Thus we have enough information to obtain δf_i. Knowing δf_i and δR gives us δf_s by (J.2). Knowing δf_s and δH gives us δT by (J.1).

J.1.3 One-Way Coupling Assumption

Ideally the change δT computed by our micro model should be fed back to the macro model. However, since the macro model is doing an implicit time step, it will be likely that this feedback from the micro model would be requested many times per macro time step. This is in practice prohibitively expensive. Instead,

we try to only compute the maximum undercooling and grain size at recalescence. So we are interested in running the micro model for only a short period of time between when the temperature has dropped below the liquidus temperature and when recalescence takes place. We then make the assumption that during this critical period of time, the macro change of enthalpy is determined by thermal gradients set up by temperature differences on a macroscopic scale and not by thermal gradients resulting from local release of latent heat. In this case, it is justified in running a standard macro solidification model and using the resulting enthalpy changes to drive the micro model. The micro model then computes temperature changes, but these temperature changes are not fed back into the macro model.

In effect, from the onset of solidification to recalescence, there are two different temperatures: T_{macro} and T. T_{macro} is not used by the micro model, except that when T_{macro} cools so that $T_{\mathrm{macro}} = T_l$, the micro model is starting with $T = T_l$ as well. The micro temperature T then is allowed to diverge from T_{macro}. T is used in the nucleation and growth model until recalescence, at which time the model has predicted the maximum undercooling $\Delta T = T_l - T$ and the corresponding grain size. At this point, the micro model is not used any more. The TRUCHAS output only contains the macro temperature T_{macro}; the micro temperature is not explicitly output. The only fields in the TRUCHAS output relating to the micro model are maximum undercooling `Max_Underc` and grain radius `Grain_R` which again do not affect the macro code in any way.

The effect of the one-way coupling assumption was investigated in [96] by comparing to a fully-coupled model. There it was found that one-way coupling causes undercoolings to be predicted accurately to within one degree celsius all along the length of a one-dimensional casting. However, the error observed was a systematic overestimation of undercooling, resulting in a systematic overestimation of nuclei density, and consequently systematic underestimation of grain size.

J.1.4 Test Problem

In the TRUCHAS test suite, the problem `grain_growth_rnt_mold.inp` tests this model. The parameters of interest appear in the `PHASE_CHANGE_PROPERTIES` namelist:

`phase_change_model:` Equal to `grain_growth_rnt` so that this model is called.

`deltempnucl:` This is ΔT_N, the undercooling where the Gaussian nucleation site density distribution peaks.

`deltempsigma:` ΔT_σ, the width of the distribution.

`ndmaxnucl:` $n_{\max}$, integral of distribution.

`a2_coeff, a3_coeff:` a_2, a_3, coefficients in cubic polynomial $v(\Delta T) = a_2(\Delta T)^2 + a_3(\Delta T)^3$ that gives tip velocity as a function of undercooling. This polynomial is a fit to the $v(\Delta T)$ function given by the Lipton model.

`forward_diffusion_coefficient:` D, solute diffusion coefficient.

`full_coupling_flag:` Set to zero, because full coupling for the Rappaz-Thévoz model is not supported.

Appendix K

Displacement, Sliding Interface and Contact Constraints

K.1 Notation

$\mathbf{u}$	-	the displacement vector for the entire domain
$\overrightarrow{u}_j$	-	the displacement vector (in ndim dimensions) for node j
$\overrightarrow{f}_j$	-	the portion of the force vector at node j that is a function of the displacement vector $\mathbf{u}$
$\overrightarrow{r}_j$	-	the portion of the force vector at node j that is a function of source terms such as thermal strain
s	-	the relative displacement of the nodes across a gap interface $\hat{n} \cdot (\overrightarrow{u}_k - \overrightarrow{u}_j)$
Λ	-	a contact function for a gap interface that depends on s . $\Lambda = 1$ for nodes in contact but not in tension, and $\Lambda = 0$ for nodes not in contact.

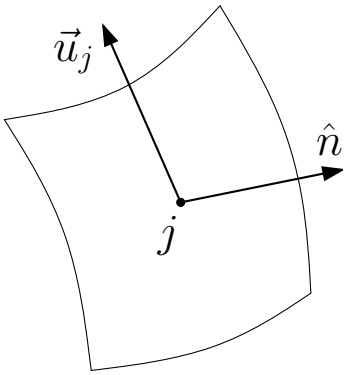
K.2 One normal displacement

For specified displacement magnitude d along surface normal $\hat{n}$

$$([I] - [\hat{n}\hat{n}^T])(\overrightarrow{f}_j + \overrightarrow{r_j}) - c[\hat{n}\hat{n}^T](\overrightarrow{u}_j - d\hat{n}) = 0 \tag{K.1}$$

or

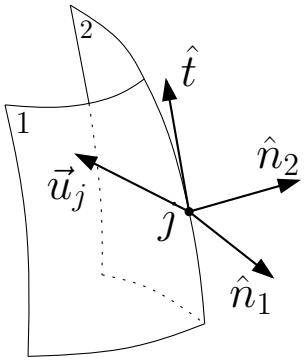
$$([I] - [\hat{n}\hat{n}^T])\,\overrightarrow{f}_j - c[\hat{n}\hat{n}^T]\,\overrightarrow{u}_j = -([I] - [\hat{n}\hat{n}^T])\,\overrightarrow{r_j} - c[\hat{n}\hat{n}^T]d\hat{n} \tag{K.2}$$



K.2.1 Preconditioning matrix

The displacement constraints are added to the preconditioning matrix by replacing the coefficients for f_j with the coefficients of the left hand side of equation K.2.

K.3 Two normal displacements



Where d_1 and d_2 are specified displacements in directions $\hat{n}_1$ and $\hat{n}_2$, and a vector along the edge between the two surface is $\hat{t} \propto \hat{n}_1 \times \hat{n}_2$,

$$[\hat{t}\hat{t}^T](\overrightarrow{f}_j + \overrightarrow{r_j}) - c([I] - [\hat{t}\hat{t}^T])(\overrightarrow{u}_j - \overrightarrow{a}) = 0 \tag{K.3}$$

or

$$[\hat{t}\hat{t}^T]\overrightarrow{f}_j - c([I] - [\hat{t}\hat{t}^T])\overrightarrow{u}_j = -[\hat{t}\hat{t}^T]\overrightarrow{r_j} - c([I] - [\hat{t}\hat{t}^T])\overrightarrow{a} \tag{K.4}$$

where the vector a is constructed as

$$\overrightarrow{a} = b_1\hat{n}_1 + b_2\hat{n}_2 \tag{K.5}$$

where

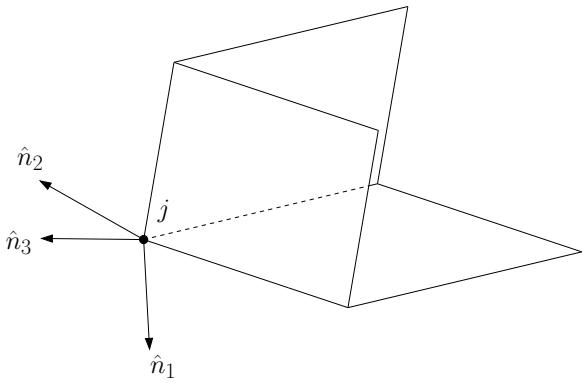
$$\begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = \frac{1}{1-\cos^2\theta} \begin{bmatrix} 1 & -\cos\theta \\ -\cos\theta & 1 \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \end{bmatrix} \tag{K.6}$$

and $\cos\theta = \hat{n}_1 \cdot \hat{n}_2$

K.3.1 Preconditioning matrix

The displacement constraints are added to the preconditioning matrix by replacing the coefficients for f_j with the coefficients of the left hand side of equation K.4.

K.4 Three normal displacements



Solve for unique $\overrightarrow{a}$ once during initialization,

$$\begin{bmatrix} \hat{n}_1 \\ \hat{n}_2 \\ \hat{n}_3 \end{bmatrix} [\overrightarrow{a}] = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \end{bmatrix} \tag{K.7}$$

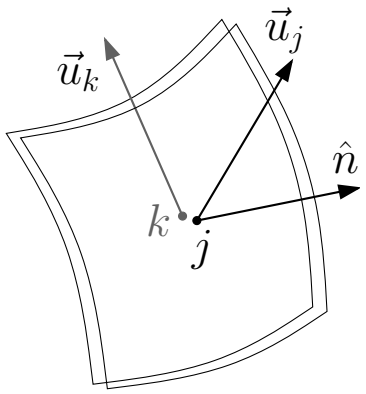
and use the trivial equation

$$c\overrightarrow{u}_j = c\overrightarrow{a} \tag{K.8}$$

K.4.1 Preconditioning matrix

The displacement constraints are added to the preconditioning matrix by replacing the coefficients for f_j with the coefficients of the left hand side of equation K.8. In this case only the diagonal coefficients are non-zero.

K.5 One normal constraint



For node j with contact function Λ :

$$([I]-\Lambda[\hat{n}\hat{n}^T])(\overrightarrow{f}_j+\overrightarrow{r_j})+\Lambda[\hat{n}\hat{n}^T](\overrightarrow{f}_j+\overrightarrow{r_j}+\overrightarrow{f}_k+\overrightarrow{r_k})+\Lambda c[\hat{n}\hat{n}^T](\overrightarrow{u}_k-\overrightarrow{u}_j)=0 \quad \text{(K.9)}$$

or

$$\overrightarrow{f}_j+\Lambda[\hat{n}\hat{n}^T](\overrightarrow{f}_k+\overrightarrow{r}_k+c(\overrightarrow{u}_k-\overrightarrow{u}_j))=-\overrightarrow{r_j} \quad \text{(K.10)}$$

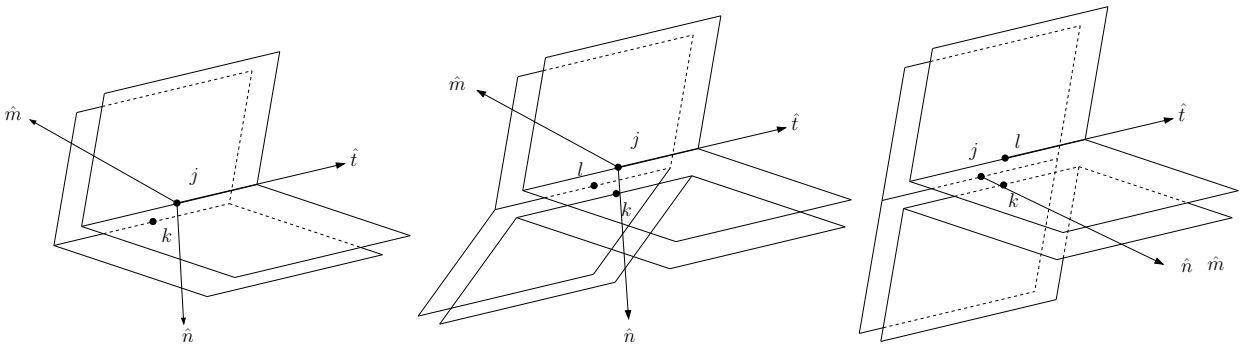
K.5.1 Preconditioning matrix

In the current version of the code normal constraints and contact are not incorporated in the preconditioning matrix.

$$\overrightarrow{f}_j-\Lambda[\hat{n}\hat{n}^T](\overrightarrow{f}_j-c(\overrightarrow{u}_k-\overrightarrow{u}_j))=-([I]-\Lambda[\hat{n}\hat{n}^T])\overrightarrow{r_j} \quad \text{(K.11)}$$

K.6 Two normal constraints

There are two normal vectors $\hat{n}$ and $\hat{m}$, one for each interface and one unit tangent vector $\hat{t} \propto \hat{n} \times \hat{m}$.



For node j there are multiple possibilities:

K.6.1 No Contact

$$\overrightarrow{f}_j = -\overrightarrow{r_j} \tag{K.12}$$

K.6.2 Contact with only one surface

As before:

gap interface 1

$$([I] - \Lambda[\hat{n}\hat{n}^T])(\overrightarrow{f}_j + \overrightarrow{r_j}) + \Lambda[\hat{n}\hat{n}^T](\overrightarrow{f}_j + \overrightarrow{r_j} + \overrightarrow{f}_k + \overrightarrow{r}_k + c(\overrightarrow{u}_k - \overrightarrow{u}_j)) = 0 \tag{K.13}$$

which is equivalent to

$$\overrightarrow{f}_j + \Lambda[\hat{n}\hat{n}^T](\overrightarrow{f}_k + \overrightarrow{r}_k + c(\overrightarrow{u}_k - \overrightarrow{u}_j)) = -\overrightarrow{r_j} \tag{K.14}$$

or gap interface 2

$$([I]-\Lambda[\hat{m}\hat{m}^T])(\overrightarrow{f}_j+\overrightarrow{r_j})+\Lambda[\hat{m}\hat{m}^T](\overrightarrow{f}_j+\overrightarrow{r_j}+\overrightarrow{f}_k+\overrightarrow{r}_k+c(\overrightarrow{u}_k-\overrightarrow{u}_j))=0 \quad \text{(K.15)}$$

which is equivalent to

$$\overrightarrow{f}_j+\Lambda[\hat{m}\hat{m}^T](\overrightarrow{f}_k+\overrightarrow{r}_k+c(\overrightarrow{u}_k-\overrightarrow{u}_j))=-\overrightarrow{r_j} \quad \text{(K.16)}$$

K.6.3 Contact with two surfaces but only one node

Interpolating between no contact and contact with both surfaces,

$$([I]-\Lambda_1\Lambda_2([I]-[\hat{t}\hat{t}^T]))(\overrightarrow{f}_j+\overrightarrow{r}_j)+\Lambda_1\Lambda_2([I]-[\hat{t}\hat{t}^T])(\overrightarrow{f}_j+\overrightarrow{r}_j+\overrightarrow{f}_k+\overrightarrow{r}_k+c(\overrightarrow{u}_k-\overrightarrow{u}_j))=0 \quad \text{(K.17)}$$

or

$$\overrightarrow{f}_j+\Lambda_1\Lambda_2([I]-[\hat{t}\hat{t}^T])(\overrightarrow{f}_k+\overrightarrow{r}_k+c(\overrightarrow{u}_k-\overrightarrow{u}_j))=-\overrightarrow{r_j} \quad \text{(K.18)}$$

Equations K.13, K.15 and K.17 can be combined to interpolate linearly between the various cases,

$$\begin{aligned}&([I]-\Lambda_1[\hat{n}\hat{n}^T]-\Lambda_2[\hat{m}\hat{m}^T]-\Lambda_1\Lambda_2([I]-[\hat{n}\hat{n}^T]-[\hat{m}\hat{m}^T]-[\hat{t}\hat{t}^T]))(\overrightarrow{f}_j+\overrightarrow{r}_j)+\\&(\Lambda_1[\hat{n}\hat{n}^T]+\Lambda_2[\hat{m}\hat{m}^T]+\Lambda_1\Lambda_2([I]-[\hat{t}\hat{t}^T]-[\hat{n}\hat{n}^T]-[\hat{m}\hat{m}^T])(\overrightarrow{f}_j+\overrightarrow{r}_j+\overrightarrow{f}_k+\overrightarrow{r}_k+c(\overrightarrow{u}_k-\overrightarrow{u}_j))=0\end{aligned} \quad \text{(K.19)}$$

or

$$\overrightarrow{f}_j+(\Lambda_1[\hat{n}\hat{n}^T]+\Lambda_2[\hat{m}\hat{m}^T]+\Lambda_1\Lambda_2([I]-[\hat{t}\hat{t}^T]-[\hat{n}\hat{n}^T]-[\hat{m}\hat{m}^T]))(\overrightarrow{f}_k+\overrightarrow{r}_k+c(\overrightarrow{u}_k-\overrightarrow{u}_j))=-\overrightarrow{r_j}$$

(K.20)

Note that if $\hat{n} = \hat{m}$ and there is only one gap node then we use equation K.10. If the angle between $\hat{n}$ and $\hat{m}$ is small, we may want to average the normals and also use equation K.10.

K.6.4 Contact with two surfaces but two different nodes

At an intersection of three different gap surfaces along an internal edge:

If Λ_{kj} is the contact function between nodes j and k,

$$([I] - \Lambda_{kj}[\hat{n}\hat{n}^T] - \Lambda_{lj}[\hat{m}\hat{m}^T] - \Lambda_{kj}\Lambda_{lj}([I] - [\hat{n}\hat{n}^T] - [\hat{m}\hat{m}^T] - [\hat{t}\hat{t}^T]))(\overrightarrow{f}_j + \overrightarrow{r}_j) +$$
$$\Lambda_{kj}[\hat{n}\hat{n}^T](\overrightarrow{f}_j + \overrightarrow{r}_j + \overrightarrow{f}_k + \overrightarrow{r}_k + c(\overrightarrow{u}_k - \overrightarrow{u}_j)) + \Lambda_{lj}[\hat{m}\hat{m}^T](\overrightarrow{f}_j + \overrightarrow{r}_j + \overrightarrow{f}_l + \overrightarrow{r}_l + c(\overrightarrow{u}_l - \overrightarrow{u}_j)) +$$
$$\Lambda_{kj}\Lambda_{lj}(([I] - [\hat{t}\hat{t}^T])(\overrightarrow{f}_j + \overrightarrow{r}_j + \overrightarrow{f}_k + \overrightarrow{r}_k + \overrightarrow{f}_l + \overrightarrow{r}_l + c(\overrightarrow{u}_k - \overrightarrow{u}_j) + c(\overrightarrow{u}_l - \overrightarrow{u}_j)) -$$
$$[\hat{n}\hat{n}^T](\overrightarrow{f}_j + \overrightarrow{r}_j + \overrightarrow{f}_k + \overrightarrow{r}_k + c(\overrightarrow{u}_k - \overrightarrow{u}_j)) - [\hat{m}\hat{m}^T](\overrightarrow{f}_j + \overrightarrow{r}_j + \overrightarrow{f}_l + \overrightarrow{r}_l + c(\overrightarrow{u}_l - \overrightarrow{u}_j))) = 0 \tag{K.21}$$

If $C_{kj} = \overrightarrow{f}_k + \overrightarrow{r}_k + c(\overrightarrow{u}_k - \overrightarrow{u}_j)$

$$\overrightarrow{f}_j + \Lambda_{kj}[\hat{n}\hat{n}^T]C_{kj} + \Lambda_{lj}[\hat{m}\hat{m}^T]C_{lj} + \Lambda_{kj}\Lambda_{lj}(([I] - [\hat{t}\hat{t}^T])(C_{kj} + C_{lj}) - [\hat{n}\hat{n}^T]C_{kj} - [\hat{m}\hat{m}^T]C_{lj}) = -\overrightarrow{r_j} \tag{K.22}$$

K.6.4.1 Two surfaces, two nodes, but only one normal

If $\hat{n} = \hat{m}$, then $\hat{t}$ is indeterminate, and we have the sum of two single constraints (equation K.10). This will be the case for a "T" intersection of three gap interfaces. (It is not currently possible to specify a "T" intersection of only two interfaces, but the equations would be the same.) In this case the last term of

equation K.20 is eliminated:

$$([I]-\Lambda_{kj}[\hat{n}\hat{n}^T]-\Lambda_{lj}[\hat{n}\hat{n}^T]+\Lambda_{kj}\Lambda_{lj}[\hat{n}\hat{n}^T])(\overrightarrow{f}_j+\overrightarrow{r}_j)+$$
$$\Lambda_{kj}[\hat{n}\hat{n}^T](\overrightarrow{f}_j+\overrightarrow{r}_j+\overrightarrow{f}_k+\overrightarrow{r}_k+c(\overrightarrow{u}_k-\overrightarrow{u}_j))+\Lambda_{lj}[\hat{n}\hat{n}^T](\overrightarrow{f}_j+\overrightarrow{r}_j+\overrightarrow{f}_l+\overrightarrow{r}_l+c(\overrightarrow{u}_l-\overrightarrow{u}_j))+$$
$$\Lambda_{kj}\Lambda_{lj}([\hat{n}\hat{n}^T](\overrightarrow{f}_j+\overrightarrow{r}_j+\overrightarrow{f}_k+\overrightarrow{r}_k+\overrightarrow{f}_l+\overrightarrow{r}_l+c(\overrightarrow{u}_k-\overrightarrow{u}_j)+c(\overrightarrow{u}_l-\overrightarrow{u}_j))-$$
$$[\hat{n}\hat{n}^T](\overrightarrow{f}_j+\overrightarrow{r}_j+\overrightarrow{f}_k+\overrightarrow{r}_k+c(\overrightarrow{u}_k-\overrightarrow{u}_j))-[\hat{n}\hat{n}^T](\overrightarrow{f}_j+\overrightarrow{r}_j+\overrightarrow{f}_l+\overrightarrow{r}_l+c(\overrightarrow{u}_l-\overrightarrow{u}_j)))=0 \tag{K.23}$$

or

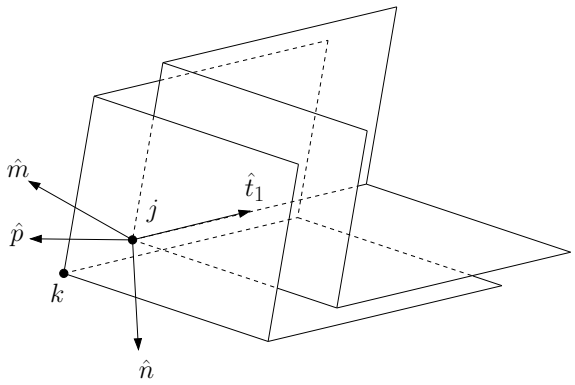
$$([I]-(\Lambda_{kj}+\Lambda_{lj}-\Lambda_{kj}\Lambda_{lj})[\hat{n}\hat{n}^T])(\overrightarrow{f}_j+\overrightarrow{r}_j)+$$
$$[\hat{n}\hat{n}^T](\Lambda_{kj}(\overrightarrow{f}_j+\overrightarrow{r}_j+\overrightarrow{f}_k+\overrightarrow{r}_k+c(\overrightarrow{u}_k-\overrightarrow{u}_j))+\Lambda_{lj}(\overrightarrow{f}_j+\overrightarrow{r}_j+\overrightarrow{f}_l+\overrightarrow{r}_l+c(\overrightarrow{u}_l-\overrightarrow{u}_j))+$$
$$\Lambda_{kj}\Lambda_{lj}((\overrightarrow{f}_j+\overrightarrow{r}_j+\overrightarrow{f}_k+\overrightarrow{r}_k+\overrightarrow{f}_l+\overrightarrow{r}_l+c(\overrightarrow{u}_k-\overrightarrow{u}_j)+c(\overrightarrow{u}_l-\overrightarrow{u}_j))-$$
$$(\overrightarrow{f}_j+\overrightarrow{r}_j+\overrightarrow{f}_k+\overrightarrow{r}_k+c(\overrightarrow{u}_k-\overrightarrow{u}_j))-(\overrightarrow{f}_j+\overrightarrow{r}_j+\overrightarrow{f}_l+\overrightarrow{r}_l+c(\overrightarrow{u}_l-\overrightarrow{u}_j)))=0 \tag{K.24}$$

or (finally)

$$\overrightarrow{f}_j+[\hat{n}\hat{n}^T](\Lambda_{kj}(\overrightarrow{f}_k+\overrightarrow{r}_k+c(\overrightarrow{u}_k-\overrightarrow{u}_j))+\Lambda_{lj}(\overrightarrow{f}_l+\overrightarrow{r}_l+c(\overrightarrow{u}_l-\overrightarrow{u}_j)))=-\overrightarrow{r}_j \tag{K.25}$$

K.6.5 Preconditioning matrix

In the current version of the code normal constraints and contact are not incorporated in the preconditioning matrix.



K.7 Three normal constraints

For node j:

- Three gap unit normals $\hat{n}\ \hat{m}\ \hat{p}$
- Three tangent unit vectors:

$$\hat{t}_1 = \frac{\hat{n}\times\hat{m}}{\|\hat{n}\times\hat{m}\|}$$
$$\hat{t}_2 = \frac{\hat{m}\times\hat{p}}{\|\hat{m}\times\hat{p}\|}$$
$$\hat{t}_3 = \frac{\hat{p}\times\hat{n}}{\|\hat{p}\times\hat{n}\|}$$

Restricted to the case where there is only one node across all three gap surfaces:

$$\begin{aligned}
&([I] - \Lambda_1[\hat{n}\hat{n}^T] - \Lambda_2[\hat{m}\hat{m}^T] - \Lambda_3[\hat{p}\hat{p}^T] + \Lambda_1\Lambda_2([\hat{t}_1\hat{t}_1^T] + [\hat{n}\hat{n}^T] + [\hat{m}\hat{m}^T] - [I]) + \\
&\Lambda_2\Lambda_3([\hat{t}_2\hat{t}_2^T] + [\hat{m}\hat{m}^T] + [\hat{p}\hat{p}^T] - [I]) + \Lambda_3\Lambda_1([\hat{t}_3\hat{t}_3^T] + [\hat{p}\hat{p}^T] + [\hat{n}\hat{n}^T] - [I]) + \\
&\Lambda_1\Lambda_2\Lambda_3(2[I] - [\hat{n}\hat{n}^T] - [\hat{m}\hat{m}^T] - [\hat{p}\hat{p}^T] - [\hat{t}_1\hat{t}_1^T] - [\hat{t}_2\hat{t}_2^T] - [\hat{t}_3\hat{t}_3^T]))(\overrightarrow{f}_j + \overrightarrow{r}_j) + \\
&(\Lambda_1[\hat{n}\hat{n}^T] + \Lambda_2[\hat{m}\hat{m}^T] + \Lambda_3[\hat{p}\hat{p}^T] + \Lambda_1\Lambda_2([I] - [\hat{t}_1\hat{t}_1^T] - [\hat{n}\hat{n}^T] - [\hat{m}\hat{m}^T]) + \\
&\Lambda_2\Lambda_3([I] - [\hat{t}_2\hat{t}_2^T] - [\hat{m}\hat{m}^T] - [\hat{p}\hat{p}^T]) + \Lambda_3\Lambda_1([I] - [\hat{t}_3\hat{t}_3^T] - [\hat{p}\hat{p}^T] - [\hat{n}\hat{n}^T]) + \\
&\Lambda_1\Lambda_2\Lambda_3(-2[I] + [\hat{n}\hat{n}^T] + [\hat{m}\hat{m}^T] + [\hat{p}\hat{p}^T] + [\hat{t}_1\hat{t}_1^T] + [\hat{t}_2\hat{t}_2^T] + [\hat{t}_3\hat{t}_3^T]))(\overrightarrow{f}_j + \overrightarrow{r}_j + \overrightarrow{f}_k + \overrightarrow{r}_k + c(\overrightarrow{u}_k - \overrightarrow{u}_j)) \\
&\qquad\qquad = 0 \quad \text{(K.26)}
\end{aligned}$$

or

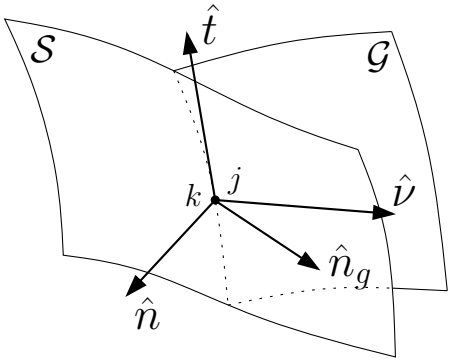
$$\overrightarrow{f}_j + (\Lambda_1[\hat{n}\hat{n}^T] + \Lambda_2[\hat{m}\hat{m}^T] + \Lambda_3[\hat{p}\hat{p}^T] + \Lambda_1\Lambda_2([I] - [\hat{t}_1\hat{t}_1^T] - [\hat{n}\hat{n}^T] - [\hat{m}\hat{m}^T]) +$$
$$\Lambda_2\Lambda_3([I] - [\hat{t}_2\hat{t}_2^T] - [\hat{m}\hat{m}^T] - [\hat{p}\hat{p}^T]) + \Lambda_3\Lambda_1([I] - [\hat{t}_3\hat{t}_3^T] - [\hat{p}\hat{p}^T] - [\hat{n}\hat{n}^T]) -$$
$$\Lambda_1\Lambda_2\Lambda_3(2[I] - [\hat{t}_1\hat{t}_1^T] - [\hat{t}_2\hat{t}_2^T] - [\hat{t}_3\hat{t}_3^T] - [\hat{n}\hat{n}^T] - [\hat{m}\hat{m}^T] - [\hat{p}\hat{p}^T]))(\overrightarrow{f}_k + \overrightarrow{r}_k + c(\overrightarrow{u}_k - \overrightarrow{u}_j))$$
$$= -\overrightarrow{r_j} \quad \text{(K.27)}$$

K.7.1 Preconditioning matrix

In the current version of the code normal constraints and contact are not incorporated in the preconditioning matrix.

K.8 One normal constraint and one normal displacement

Intersection between a gap interface and a surface with a displacement constraint:



For node j:

- Surface unit normal $\hat{n}$
- Gap unit normal $\hat{n}_g$
- Tangent unit vector $\hat{t} = \frac{\hat{n}\times\hat{n}_g}{\|\hat{n}\times\hat{n}_g\|}$
- Vector to complete the orthogonal set $\hat{v} = \hat{t} \times \hat{n}$

- $\cos\theta = \hat{v}\cdot\hat{n}_g$

$$(([I]-[\hat{n}\hat{n}^T])-\Lambda([I]-[\hat{n}\hat{n}^T]-[\hat{t}\hat{t}^T]))(\overrightarrow{f}_j+\overrightarrow{r}_j)+$$
$$\Lambda[\hat{v}\hat{v}^T](\overrightarrow{f}_j+\overrightarrow{r}_j+\overrightarrow{f}_k+\overrightarrow{r}_k)+\Lambda c\cos^2\theta[\hat{v}\hat{v}^T](\overrightarrow{u}_k-\overrightarrow{u}_j)-c[\hat{n}\hat{n}^T](\overrightarrow{u}_j-d\hat{n})=0 \quad \text{(K.28)}$$

or (using $[\hat{n}\hat{n}^T]+[\hat{v}\hat{v}^T]+[\hat{t}\hat{t}^T]=[I]$):

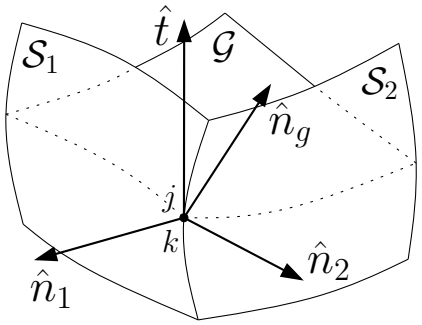
$$([I]-[\hat{n}\hat{n}^T])\,\overrightarrow{f}_j+\Lambda[\hat{v}\hat{v}^T](\overrightarrow{f}_k+\overrightarrow{r}_k+c\cos^2\theta(\overrightarrow{u}_k-\overrightarrow{u}_j))-c[\hat{n}\hat{n}^T]\,\overrightarrow{u}_j=-([I]-[\hat{n}\hat{n}^T])\,\overrightarrow{r}_j-c[\hat{n}\hat{n}^T]d\hat{n} \quad \text{(K.29)}$$

K.8.1 Preconditioning matrix

In the current version of the code normal constraints and contact are not incorporated in the preconditioning matrix. However, the normal displacement constraint is included in the same way as for a normal displacement without the interface constraints.

K.9 Two displacements, one normal constraint

Intersection between a gap interface and a surface with a displacement constraint:



- Surface unit normals $\hat{n}_1$ and $\hat{n}_2$

- Gap unit normal $\hat{n}_g$
- Tangent unit vector $\hat{t} = \frac{\hat{n}_1 \times \hat{n}_2}{\|\hat{n}_1 \times \hat{n}_2\|}$
- $\cos\theta = \hat{t} \cdot \hat{n}_g$
- d_1 and d_2 are specified displacements in directions $\hat{n}_1$ and $\hat{n}_2$

$$\overrightarrow{a} = b_1 \hat{n}_1 + b_2 \hat{n}_2 \tag{K.30}$$

where

$$\begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = \frac{1}{1 - \cos^2\theta} \begin{bmatrix} 1 & -\cos\theta \\ -\cos\theta & 1 \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \end{bmatrix} \tag{K.31}$$

For node j:

$$(1-\Lambda)[\hat{t}\hat{t}^T](\overrightarrow{f}_j + \overrightarrow{r}_j) + \Lambda[\hat{t}\hat{t}^T](\overrightarrow{f}_j + \overrightarrow{r}_j + \overrightarrow{f}_k + \overrightarrow{r}_k + c\cos^2\theta(\overrightarrow{u}_k - \overrightarrow{u}_j)) - c([I] - [\hat{t}\hat{t}^T])(\overrightarrow{u}_j - \overrightarrow{a}) = 0 \tag{K.32}$$

or

$$[\hat{t}\hat{t}^T]\overrightarrow{f}_j + \Lambda[\hat{t}\hat{t}^T](\overrightarrow{f}_k + \overrightarrow{r}_k + c\cos^2\theta(\overrightarrow{u}_k - \overrightarrow{u}_j)) - c([I] - [\hat{t}\hat{t}^T])\overrightarrow{u}_j = -[\hat{t}\hat{t}^T]\overrightarrow{r}_j - c([I] - [\hat{t}\hat{t}^T])\overrightarrow{a} \tag{K.33}$$

K.9.1 Preconditioning matrix

In the current version of the code normal constraints and contact are not incorporated in the preconditioning matrix. However, the normal displacement constraint is included in the same way as for a two normal displacements without the interface constraint.

K.10 One displacement, two normal constraints

- Surface normal $\hat{n}$
- d is the magnitude of the specified displacement.

- gap normal vectors $\hat{m}$ and $\hat{p}$, one for each interface
- two unit tangent vectors: $\hat{t}_1 = \frac{\hat{n}\times\hat{m}}{\|\hat{n}\times\hat{m}\|}$ and $\hat{t}_2 = \frac{\hat{n}\times\hat{p}}{\|\hat{n}\times\hat{p}\|}$
- Vectors $\hat{v}$ and $\hat{w}$ are defined to make two orthogonal sets with $\hat{n}$: $\hat{v} = \hat{t}_1 \times \hat{n}$ and $\hat{w} = \hat{t}_2 \times \hat{n}$
- $\cos\theta_1 = \hat{v}\cdot\hat{m}$ and $\cos\theta_2 = \hat{w}\cdot\hat{p}$

For node j there are multiple possibilities:

K.10.1 No Contact

$$([I] - [\hat{n}\hat{n}^T])\,\overrightarrow{f}_j - c[\hat{n}\hat{n}^T]\,\overrightarrow{u}_j = -([I] - [\hat{n}\hat{n}^T])\,\overrightarrow{r_j} - c[\hat{n}\hat{n}^T]d\hat{n} \tag{K.34}$$

K.10.2 Contact with only one surface

As before:

gap interface 1

$$([I]-[\hat{n}\hat{n}^T]-\Lambda_1([I]-[\hat{n}\hat{n}^T]-[\hat{t_1}\hat{t_1}^T]))(\overrightarrow{f}_j+\overrightarrow{r}_j)+\Lambda_1[\hat{v}\hat{v}^T](\overrightarrow{f}_j+\overrightarrow{r}_j+\overrightarrow{f}_k+\overrightarrow{r}_k+\cos^2\theta_1 c(\overrightarrow{u}_k-\overrightarrow{u}_j))$$
$$- c[\hat{n}\hat{n}^T](\overrightarrow{u}_j - d\hat{n}) = 0 \tag{K.35}$$

Using $([I] - [\hat{n}\hat{n}^T] - [\hat{t_1}\hat{t_1}^T]) = [\hat{v}\hat{v}^T]$ we obtain:

$$([I]-[\hat{n}\hat{n}^T])\,\overrightarrow{f}_j+[\hat{v}\hat{v}^T](\overrightarrow{f}_k+\overrightarrow{r}_k+\cos^2\theta_1 c(\overrightarrow{u}_k-\overrightarrow{u}_j))-c[\hat{n}\hat{n}^T]\,\overrightarrow{u}_j = -([I]-[\hat{n}\hat{n}^T])\,\overrightarrow{r_j}-c[\hat{n}\hat{n}^T]d\hat{n} \tag{K.36}$$

or gap interface 2

$$([I]-[\hat{n}\hat{n}^T])\,\overrightarrow{f}_j+[\hat{w}\hat{w}^T](\overrightarrow{f}_k+\overrightarrow{r}_k+\cos^2\theta_2 c(\overrightarrow{u}_k-\overrightarrow{u}_j))c[\hat{n}\hat{n}^T]\,\overrightarrow{u}_j = -([I]-[\hat{n}\hat{n}^T])\,\overrightarrow{r_j}-c[\hat{n}\hat{n}^T]d\hat{n}$$

(K.37)

K.10.3 Contact with two surfaces but only one node

If $\Lambda_1 = \Lambda_2 = 1$

$$([I]-[\hat{n}\hat{n}^T])(\overrightarrow{f}_j+\overrightarrow{r_j}+\overrightarrow{f}_k+\overrightarrow{r}_k+(\cos^2\theta_1+\cos^2\theta_2)c(\overrightarrow{u}_k-\overrightarrow{u}_j))-c[\hat{n}\hat{n}^T]\overrightarrow{u}_j = -c[\hat{n}\hat{n}^T]d\hat{n} \tag{K.38}$$

Equations K.36, K.37 and K.38 can be combined with contact functions for each interface Λ_1 and Λ_2,

$$\begin{aligned}
&([I]-[\hat{n}\hat{n}^T]-\Lambda_1([I]-[\hat{n}\hat{n}^T]-[\hat{t_1}\hat{t_1}^T])-\Lambda_2([I]-[\hat{n}\hat{n}^T]-[\hat{t_2}\hat{t_2}^T])+\Lambda_1\Lambda_2([I]-[\hat{n}\hat{n}^T]-[\hat{t_1}\hat{t_1}^T]-[\hat{t_2}\hat{t_2}^T]))(\overrightarrow{f}_j+\overrightarrow{r}_j\\
&\Lambda_1[\hat{v}\hat{v}^T](\overrightarrow{f}_j+\overrightarrow{r}_j+\overrightarrow{f}_k+\overrightarrow{r}_k+c\cos^2\theta_1(\overrightarrow{u}_k-\overrightarrow{u}_j))+\Lambda_2[\hat{w}\hat{w}^T](\overrightarrow{f}_j+\overrightarrow{r}_j+\overrightarrow{f}_k+\overrightarrow{r}_k+c\cos^2\theta_2(\overrightarrow{u}_k-\overrightarrow{u}\\
&\Lambda_1\Lambda_2(([I]-[\hat{n}\hat{n}^T])(\overrightarrow{f}_j+\overrightarrow{r}_j+\overrightarrow{f}_k+\overrightarrow{r}_k+c(\cos^2\theta_1+\cos^2\theta_2)(\overrightarrow{u}_k-\overrightarrow{u}_j))-[\hat{v}\hat{v}^T](\overrightarrow{f}_j+\overrightarrow{r}_j+\overrightarrow{f}_k+\overrightarrow{r}_k+\\
&\quad[\hat{w}\hat{w}^T](\overrightarrow{f}_j+\overrightarrow{r}_j+\overrightarrow{f}_k+\overrightarrow{r}_k+c\cos^2\theta_2(\overrightarrow{u}_k-\overrightarrow{u}_j)))-[\hat{n}\hat{n}^T]c(\overrightarrow{u}_j-d\hat{n}) = 0
\end{aligned} \tag{K.39}$$

using the identities $[\hat{n}\hat{n}^T]+[\hat{t_1}\hat{t_1}^T]+[\hat{v}\hat{v}^T]=[I]$ and $[\hat{n}\hat{n}^T]+[\hat{t_2}\hat{t_2}^T]+[\hat{w}\hat{w}^T]=[I]$,

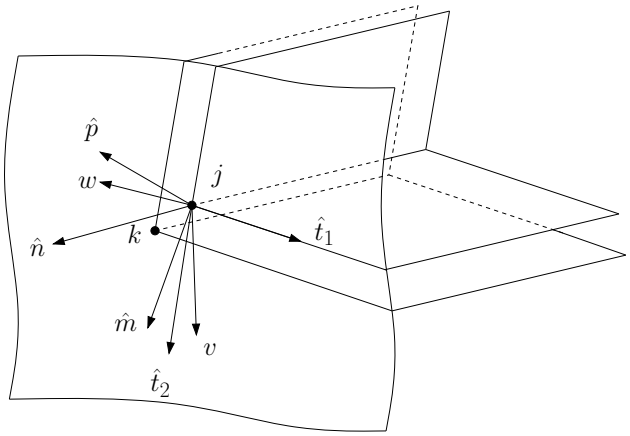
$$\begin{aligned}
&([I]-[\hat{n}\hat{n}^T]-\Lambda_1[\hat{v}\hat{v}^T]-\Lambda_2[\hat{w}\hat{w}^T]+\Lambda_1\Lambda_2([\hat{v}\hat{v}^T]+[\hat{w}\hat{w}^T]-([I]-[\hat{n}\hat{n}^T])))(\overrightarrow{f}_j+\overrightarrow{r_j})+\\
&\Lambda_1[\hat{v}\hat{v}^T](\overrightarrow{f}_j+\overrightarrow{r}_j+\overrightarrow{f}_k+\overrightarrow{r}_k+c\cos^2\theta_1(\overrightarrow{u}_k-\overrightarrow{u}_j))+\Lambda_2[\hat{w}\hat{w}^T](\overrightarrow{f}_j+\overrightarrow{r}_j+\overrightarrow{f}_k+\overrightarrow{r}_k+c\cos^2\theta_2(\overrightarrow{u}_k-\overrightarrow{u}\\
&\Lambda_1\Lambda_2(([I]-[\hat{n}\hat{n}^T])(\overrightarrow{f}_j+\overrightarrow{r}_j+\overrightarrow{f}_k+\overrightarrow{r}_k+c(\cos^2\theta_1+\cos^2\theta_2)(\overrightarrow{u}_k-\overrightarrow{u}_j))-[\hat{v}\hat{v}^T](\overrightarrow{f}_j+\overrightarrow{r}_j+\overrightarrow{f}_k+\overrightarrow{r}_k+\\
&\quad[\hat{w}\hat{w}^T](\overrightarrow{f}_j+\overrightarrow{r}_j+\overrightarrow{f}_k+\overrightarrow{r}_k+c\cos^2\theta_2(\overrightarrow{u}_k-\overrightarrow{u}_j)))-[\hat{n}\hat{n}^T]c(\overrightarrow{u}_j-d\hat{n}) = 0
\end{aligned} \tag{K.40}$$

or

$$([I]-[\hat{n}\hat{n}^T])\overrightarrow{f}_j+(\Lambda_1[\hat{v}\hat{v}^T]+\Lambda_2[\hat{w}\hat{w}^T]+\Lambda_1\Lambda_2([I]-[\hat{n}\hat{n}^T]-[\hat{v}\hat{v}^T]-[\hat{w}\hat{w}^T]))(\overrightarrow{f}_k+\overrightarrow{r}_k)+$$
$$(\Lambda_1[\hat{v}\hat{v}^T]\cos^2\theta_1+\Lambda_2[\hat{w}\hat{w}^T]\cos^2\theta_2+\Lambda_1\Lambda_2(([I]-[\hat{n}\hat{n}^T])(\cos^2\theta_1+\cos^2\theta_2)-$$
$$[\hat{v}\hat{v}^T]\cos^2\theta_1-[\hat{w}\hat{w}^T]\cos^2\theta_2))(c(\overrightarrow{u}_k-\overrightarrow{u}_j))-c[\hat{n}\hat{n}^T]\overrightarrow{u}_j$$
$$=-([I]-[\hat{n}\hat{n}^T])\overrightarrow{r}_j-c[\hat{n}\hat{n}^T]d\hat{n} \quad \text{(K.41)}$$

Note that if $\hat{m}=\hat{p}$ and there is only one gap node then we use equation K.29. If the angle between $\hat{m}$ and $\hat{p}$ is small, we may want to average the normals and also use equation K.29.

K.10.4 Contact with two surfaces but two different nodes



At an intersection of three different gap surfaces along an internal edge and a normal constraint at the free surface (after equation K.40):

$$([I]-[\hat{n}\hat{n}^T]-\Lambda_1[\hat{v}\hat{v}^T]-\Lambda_2[\hat{w}\hat{w}^T]+\Lambda_1\Lambda_2([\hat{v}\hat{v}^T]+[\hat{w}\hat{w}^T]-([I]-[\hat{n}\hat{n}^T])))(\overrightarrow{f}_j+\overrightarrow{r_j})+$$
$$\Lambda_1[\hat{v}\hat{v}^T](\overrightarrow{f}_j+\overrightarrow{r}_j+\overrightarrow{f}_k+\overrightarrow{r}_k+c\cos^2\theta_1(\overrightarrow{u}_k-\overrightarrow{u}_j))+\Lambda_2[\hat{w}\hat{w}^T](\overrightarrow{f}_j+\overrightarrow{r}_j+\overrightarrow{f}_l+\overrightarrow{r}_l+c\cos^2\theta_2(\overrightarrow{u}_l-\overrightarrow{u}$$
$$\Lambda_1\Lambda_2(([I]-[\hat{n}\hat{n}^T])(\overrightarrow{f}_j+\overrightarrow{r}_j+\overrightarrow{f}_k+\overrightarrow{r}_k+\overrightarrow{f}_l+\overrightarrow{r}_l+c\cos^2\theta_1(\overrightarrow{u}_k-\overrightarrow{u}_j)+c\cos^2\theta_2(\overrightarrow{u}_l-\overrightarrow{u}_j))-[\hat{v}\hat{v}^T](\overrightarrow{}$$
$$[\hat{w}\hat{w}^T](\overrightarrow{f}_j+\overrightarrow{r}_j+\overrightarrow{f}_l+\overrightarrow{r}_l+c\cos^2\theta_2(\overrightarrow{u}_l-\overrightarrow{u}_j)))-[\hat{n}\hat{n}^T]c(\overrightarrow{u}_j-d\hat{n})=0$$
(K.42)

If $C_{kj} = \overrightarrow{f}_k + \overrightarrow{r}_k + \cos^2\theta_1 c(\overrightarrow{u}_k - \overrightarrow{u}_j)$, $C_{lj} = \overrightarrow{f}_l + \overrightarrow{r}_l + \cos^2\theta_2 c(\overrightarrow{u}_l - \overrightarrow{u}_j)$, and Λ_{kj} and Λ_{lj} are the contact functions between nodes j and k, and l and j respectively:

$$
\begin{aligned}
&([I] - [\hat{n}\hat{n}^T])\,\overrightarrow{f}_j + \Lambda_{kj}[\hat{v}\hat{v}^T]C_{kj} + \Lambda_{lj}[\hat{w}\hat{w}^T]C_{lj} + \\
&\Lambda_{kj}\Lambda_{lj}(([I] - [\hat{n}\hat{n}^T])(C_{kj} + C_{lj}) - [\hat{v}\hat{v}^T]C_{kj} - [\hat{w}\hat{w}^T]C_{lj}) - c[\hat{n}\hat{n}^T]\,\overrightarrow{u}_j \\
&\qquad = -([I] - [\hat{n}\hat{n}^T])\,\overrightarrow{r_j} - c[\hat{n}\hat{n}^T]d\hat{n} \qquad \text{(K.43)}
\end{aligned}
$$

K.10.4.1 Two surfaces, two nodes, but only one normal

If $\hat{m} = \hat{p}$, then sliding along the two surfaces should be allowed and we have the sum of two single constraints (equation K.29). This will be the case for a "T" intersection of three gap interfaces. (It is not currently possible to specify a "T" intersection of only two interfaces, but the equations would be the same.) In this case the last term of equation K.43 is not used:

$$
([I]-[\hat{n}\hat{n}^T])\,\overrightarrow{f}_j+\Lambda_{kj}[\hat{v}\hat{v}^T]C_{kj}+\Lambda_{lj}[\hat{w}\hat{w}^T]C_{lj}-c[\hat{n}\hat{n}^T]\,\overrightarrow{u}_j = -([I]-[\hat{n}\hat{n}^T])\,\overrightarrow{r_j} - c[\hat{n}\hat{n}^T]d\hat{n} \qquad \text{(K.44)}
$$

K.10.5 Preconditioning matrix

In the current version of the code normal constraints and contact are not incorporated in the preconditioning matrix. However, the normal displacement constraint is included in the same way as for a normal displacement without the interface constraints.

Bibliography

[1] D. B. Kothe and R. C. Mjolsness. RIPPLE: A new model for incompressible flows with free surfaces. *AIAA Journal*, 30:2694–2700, 1992.

[2] W.J. Rider and D.B. Kothe. Reconstructing volume tracking. *Journal of Computational Physics*, 141(2):112–152, 1998.

[3] J. U. Brackbill, D. B. Kothe, and C. Zemach. A continuum method for modeling surface tension. *Journal of Computational Physics*, 100:335–354, 1992.

[4] Y.D. Fryer, C. Bailey, M. Cross, and C.H. Lai. A control volume procedure for solving the elastic stress-strain equations on an unstructured mesh. *Applied Mathematical Modelling*, 15:639–645, 1991.

[5] C. Bailey and M. Cross. A finite volume procedure to solve elastic solid mechanics problems in three dimensions on an unstructured mesh. *International Journal for Numerical Methods in Engineering*, 38:1757–1776, 1995.

[6] C. M. Rhie and W. L. Chow. A numerical study of the turbulent flow past an isolated airfoil with trailing edge separation. *AIAA Journal*, 21:1525–1532, 1983.

[7] D. Kothe, D. Juric, K. Lam, and B. Lally. Numerical recipes for mold filling simulation. In *Modeling of Casting, Welding, and Advanced Solidification Processes VIII*, New York, 1998.[1] TMS Publishers.

[8] D. B. Kothe. Perspective on Eulerian finite volume methods for incompressible interfacial flows. In H. Kuhlmann and H. Rath, editors, *Free Surface Flows*, pages 267–331, New York, NY, 1998. Springer-Verlag.

[9] W. J. Rider and D. B. Kothe. Stretching and tearing interface tracking methods. Technical Report AIAA 95–1717, AIAA, 1995.[2] Presented at the 12th AIAA CFD Conference.

[10] W. J. Rider and D. B. Kothe. Reconstructing volume tracking. *Journal of Computational Physics*, 141:112–152, 1998.

[1] Available at `http://www.lanl.gov/telluride`.
[2] Available at `http://www.lanl.gov/telluride`.

[11] S. J. Mosso, B. K. Swartz, D. B. Kothe, and R. C. Ferrell. A parallel, volume-tracking algorithm for unstructured meshes. In P. Schiano, A. Ecer, J. Periaux, and N. Satofuka, editors, *Parallel Computational Fluid Dynamics: Algorithms and Results Using Advanced Computers*, pages 368–375, Capri, Italy, 1997.[3] Elsevier Science.

[12] D. B. Kothe, W. J. Rider, S. J. Mosso, J. S. Brock, and J. I. Hochstein. Volume tracking of interfaces having surface tension in two and three dimensions. Technical Report AIAA 96–0859, AIAA, 1996.[4] Presented at the 34rd Aerospace Sciences Meeting and Exhibit.

[13] D.B. Kothe, M.W. Williams, K.L. Lam, D.R. Korzekwa, P.K. Tubesing, and E.G. Puckett. A second-order accurate, linearity-preserving volume tracking algorithm for free surface flows on 3-D unstructured meshes. In *Proceedings of the 3rd ASME/JSME Joint Fluids Engineering Conference*, 1999.

[14] M. W. Williams, D. B. Kothe, and E. G. Puckett. Approximating interface topologies with applications to interface tracking algorithms. Technical Report 99–1076, AIAA, 1999. Presented at the 37th Aerospace Sciences Meeting.

[15] D. L. Youngs. Time-dependent multi-material flow with large fluid distortion. In K. W. Morton and M. J. Baines, editors, *Numerical Methods for Fluid Dynamics*, pages 273–285, 1982.

[16] D. B. Kothe. PAGOSA: A massively-parallel, multi-material hydrodynamics model for three-dimensional high-speed flow and high-rate deformation. Technical Report LA–UR–92–4306, Los Alamos National Laboratory, 1992.

[17] M. M. Francois, S. J. Cummins, E. D. Dendy, D. B. Kothe, J. M. Sicilian, and M. W. Williams. A balanced-force algorithm for continuous and sharp interfacial surface tension models within a volume tracking framework. *Journal of Computational Physics*, 2005. to appear, also Los Alamos Technical Report LA-UR-05-0674.

[18] M. Rudman. A volume-tracking method for incompressible multifluid flows with large density variations. *International Journal for Numerical Methods in Fluids*, 28:357–378, 1998.

[19] M. W. Williams. *Numerical Methods for Tracking Interfaces with Surface Tension in 3-D mold filling processes*. PhD thesis, University of California, Davis, 2000.

[20] T. J. Barth. Recent developments in high order K-exact reconstruction on unstructured meshes. Technical Report AIAA–93–0668, AIAA, 1993. Presented at the 31st Aerospace Sciences Meeting and Exhibit.

[21] P.C. Carman. Fluid flow through granular beds. *Trans. Inst. Chem. Engrs.*, 15:150–166, 1937.

[22] T. W. Clyne and W. Kurz. Solute redisribution during solidification with rapid solid state diffusion. *Metallurgical Transactions A*, 12:965–971, 1981.

[3] Available at `http://www.lanl.gov/telluride`.
[4] Available at `http://www.lanl.gov/telluride`.

[23] D. A. Knoll, D. B. Kothe, and B. Lally. A new nonlinear solution method for phase change problems. *Numerical Heat Transfer, Part B*, 35:436–459, 1999.

[24] P.S. Follansbee and U.F. Kocks. A constitutive description of the deformation of copper based on the use of the mechanical threshold stress as an internal state variable. *Acta Metallurgica*, 36:81–93, 1988.

[25] O. C. Zienkiewicz. *The Finite Element Method*. McGraw-Hill, New York, NY, 1977.

[26] A. Bossavit. *Computational Electromagnetism: Variational formulations, Complementarity, Edge Elements*. Academic Press, San Diego, CA, 1998.

[27] R. Hiptmair. Multigrid method for Maxwell's equations. *SIAM Journal on Numerical Analysis*, 36(1):204–225, 1998.

[28] J.P. Hayes. *Computer Architecture and Organization*. McGraw-Hill, New York, second edition, 1988.

[29] W.D. Hillis and G.L. Steele. Data parallel algorithms. *Communications of the ACM*, 29(12):1170–1183, 1986. Available on-line at http://portal.acm.org/citation.cfm?doid=7902.7903.

[30] See *http://www.mpi-forum.org/*.

[31] M. Snir, S.W. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra. *MPI: The Complete Reference*. The MIT Press, Cambridge, MA, 1996.

[32] E. Lusk W. Gropp and A. Skjellum. *Using MPI: Portable Parallel Programming with the Message Passing Interface*. The MIT Press, Cambridge, MA, 1994.

[33] The PGSLib package, now named Diablo95, is available at *http://diablo95.sourceforge.net/*.

[34] T. J. Barth. Aspects of unstructured grids and finite-volume solvers for Euler and Navier-Stokes equations, 1995. VKI/NASA/AGARD Special Course on Unstructured Grid Methods for Advection Dominated Flows, AGARD Publication R-787.

[35] Carl F. Ollivier-Gooch. Quasi-eno schemes for unstructured meshes based on unlimited data-dependent least-squares reconstruction. *Journal of Computational Physics*, 133:6–17, 1997.

[36] M. Shashkov J. Hyman, J. Morel and S. Steinberg. Mimetic finite difference methods for diffusion equations. *Computational Geosciences*, 6:333–352, 2002.

[37] Jim E. Morel, Michael L. Hall, and Mikhail J. Shashkov. A Local Support-Operators Diffusion Discretization Scheme for Hexahedral Meshes. *Journal of Computational Physics*, 170(1):338–372, June 2001. LA-UR–99-4358. Available online at `http://www.LANL.gov/Augustus`.

[38] Michael L. Hall and Jim E. Morel. Diffusion Discretization Schemes in Augustus: A New Hexahedral Symmetric Support Operator Method. In *Proceedings of the 1998 Nuclear Explosives Code Developers Conference (NECDC)*, Las Vegas, NV, October 25–30 1998. LA-UR–98-3146. Available online at `http://www.LANL.gov/Augustus`.

[39] Michael L. Hall, Jim E. Morel, and Mikhail J. Shashkov. A Local Support Operator Diffusion Discretization Scheme for Hexahedral Meshes. Technical Report LA-UR–99-5834, Los Alamos National Laboratory, October 21 1999. JOWOG 42 Presentation. Available online at http://www.LANL.gov/Augustus.

[40] Markus Berndt, Konstantin Lipnikov, Mikhail Shashkov, Mary F. Wheeler, and Ivan Yotov. Superconvergence of the velocity in mimetic finite difference methods on quadrilaterals. *SIAM Journal on Numerical Analysis*, 43(4):1728–1749, 2005.

[41] R. Varga. *Matrix Iterative Analysis*. Prentice-Hall, Englewood Cliffs, NJ, 1962.

[42] G. Strang. *Linear Algebra and Its Applications*. Harcourt Brace Jovanovich, 1976.

[43] G. Strang. *Introduction to Applied Mathematics*. Wellesley-Cambridge, 1986.

[44] G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, 1989.

[45] Y. Saad. *Iterative Methods for Sparse Linear Systems*. PWS Publishing Company, 1996.

[46] R. Barrett et al. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. SIAM, Philadelphia, PA, 1993.

[47] J. R. Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Technical Report CMU-CS-94-125, Carnegie Mellon University, 1994.

[48] W. Schönauer and R. Weiss. An engineering approach to generalized conjugate gradient methods and beyond. *Applied Numerical Mathematics*, 19:175–206, 1995.

[49] A. Meister. Comparison of different Krylov subspace methods embedded in an implicit finite volume scheme for the computation of viscous and inviscid flow fields on unstructured grids. *Journal of Computational Physics*, 140:311–345, 1998.

[50] A. Brandt. Multi-level adaptive solutions to boundary-value problems. *Mathematics of Computation*, 31:333–390, 1977.

[51] W. L. Briggs. *A Multigrid Tutorial*. SIAM, Philadelphia, PA, 1987.

[52] P. Wesseling. *An Introduction to Multigrid Methods*. Wiley, 1992.

[53] W. Shyy, S. S. Thakur, H. Ouyang, J. Liu, and E. Blosch. *Computational Techniques for Complex Transport Phenomena*. Cambridge University Press, 1997.

[54] W. J. Rider. Approximate projection methods for incompressible flow: Implementation, variants and robustness. Technical Report LA–UR–94–2000, Los Alamos National Laboratory, 1994.

[55] W. J. Rider, D. B. Kothe, E. G. Puckett, and I. D. Aleinov. Accurate and robust methods for variable density incompressible flows with discontinuities. In V. Venkatakrishnan, M. D. Salas, and S. R. Chakravarthy, editors, *Workshop on Barriers and Challenges in Computational Fluid Dynamics*, pages 213–230, Boston, MA, 1998.[5] Kluwer Academic Publishers.

[5] Available at http://www.lanl.gov/telluride.

[56] C. Liu, Z. Liu, and S. McCormick. An efficient multigrid scheme for elliptic equations with discontinuous coefficients. *Communications in Applied Numerical Methods*, 8:621–631, 1992.

[57] R. Kettler and J. A. Meijerink. A multigrid method and a combined multigrid-conjugate gradient method for elliptic problems with strongly discontinuous coefficients. Technical Report 604, Shell Corporation, Rijswijk, The Netherlands, 1981.

[58] O. Tatebe. The multigrid preconditioned conjugate gradient method. In N. D. Melson, T. A. Manteuffel, and S. F. McCormick, editors, *Proceedings of the Sixth Copper Mountain Conference on Multigrid Methods*, pages 621–634, Copper Mountain, CO, 1993.

[59] E. G. Puckett, A. S. Almgren, J. B. Bell, D. L. Marcus, and W. J. Rider. A second-order projection method for tracking fluid interfaces in variable density incompressible flows. *Journal of Computational Physics*, 130:269–282, 1997.

[60] J. C. Meza and R. S. Tuminaro. A multigrid preconditioner for the semiconductor equations. *SIAM Journal on Scientific Computing*, 17:118–132, 1996.

[61] S. F. Ashby and R. D. Falgout. A parallel multigrid preconditioned conjugate gradient algorithm for groundwater flow simulations. *Nuclear Science and Engineering*, 124:145–159, 1996.

[62] B. Lally, R. Ferrell, D. Knoll, D. Kothe, and J. Turner. Parallel two-level additive-Schwarz preconditioning on 3D unstructured meshes for solution of solidification problems. In T. Manteuffel and S. McCormick, editors, *Proceedings of the Eighth Copper Mountain Conference on Iterative Methods*, Copper Mountain, CO, 1998.[6]

[63] X.-C. Cai. A family of overlapping Schwarz algorithms for nonsymmetric and indefinite elliptic problems. In D. E. Keyes, Y. Saad, and D. G. Truhlar, editors, *Domain-Based Parallelism and Problem Decomposition Methods in Computational Science and Engineering*, pages 1–19, Philadelphia, PA, 1995. SIAM.

[64] D. B. Kothe et al. The Telluride Project. For information, see *http://www.lanl.gov/telluride* or contact the project team at *telluride@lanl.gov*.

[65] D. B. Kothe, R. C. Ferrell, J. A. Turner, and S. J. Mosso. A high resolution finite volume method for efficient parallel simulation of casting processes on unstructured meshes. In *Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing*, Minneapolis, MN, March 14–17, 1997.[7]

[66] J.E. Morel, J.E. Dendy, M.L. Hall, and S.W. White. A cell-centered Lagrangian-mesh diffusion differencing scheme. *Journal of Computational Physics*, 103:286, 1992.

[67] D. J. Mavripilis and V. Venkatakrishnan. A 3D agglomeration multigrid solver for the Reynolds-averaged Navier-Stokes equations on unstructured meshes. *Intl. J. Numer. Meths. Fluids*, 23:527–544, 1996.

[6] Available at `http://www.lanl.gov/telluride`.
[7] Available at `http://www.lanl.gov/telluride`.

[68] Barry Smith, Petter Bjorstad, and William Gropp. *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge University Press1, Cambridge, 1996.

[69] D. B. Kothe and W. J. Rider. Constrained minimization for monotonic reconstruction. In D. Kwak, editor, *Proceedings of the Thirteenth AIAA Computational Fluid Dynamics Conference*, pages 955–964, 1997.[8] AIAA Paper 97–2036.

[70] Y. Saad and M.H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymetric linear systems. *SIAM J. Sci. Stat. Comput.*, 7:856, 1986.

[71] F. de la Vallee Poussin. An accelerated relaxation algorithm for the iterative solution of elliptic equations. *SIAM Journal on Numerical Analysis*, 2:340–351, 1968.

[72] A. Settari and K. Aziz. A generalization of the additive correction methods for the iterative solution of matrix equations. *SIAM J. Numer. Anal.*, 10:506–521, 1973.

[73] R.A. Nicolaides. On multiple grid and related techniques for solving discrete elliptic systems. *J. Comput. Phys.*, 19:418–431, 1975.

[74] J. A. Turner, R. C. Ferrell, and D. B. Kothe. JTpack90: A parallel, object-based Fortran 90 linear algebra package. In *Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing*, Minneapolis, MN, March 14–17, 1997.[9]

[75] R. C. Ferrell, D. B. Kothe, and J. A. Turner. PGSLib: A library for portable, parallel, unstructured mesh simulations. In *Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing*, Minneapolis, MN, March 14–17, 1997.[10]

[76] D.S. Kershaw. The incomplete Cholesky-conjugate gradient method for the iterative solution of systems of linear equations. *Journal of Computational Physics*, 26:43–65, 1978.

[77] R.W. Freund. A transpose-free quasi-minimal residual algorithm for non-hermitian linear systems. *SIAM J. Sci. Comput.*, 14:470–482, 1993.

[78] H. A. Van der Vorst. Bi-CGSTAB:a fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 13:631–644, 1992.

[79] J.E. Dendy. Black box multigrid. *Journal of Computational Physics*, 48:366, 1982.

[80] D. A. Knoll and W. J. Rider. A multigrid preconditioned Newton-Krylov method. Technical Report LA–UR–97–4013, Los Alamos National Laboratory, 1998. Submitted for publication to the *SIAM Journal of Scientific Computing*.

[81] D.A. Knoll, G. Lapenta, and J.U. Brackbill. A multilevel field solver for implicit kinetic, plasma simulation. *Los Alamos National Laboratory Report LA-UR-98-2159, to appear in J. Comput. Phys.*, 1998.

[8] Available at `http://www.lanl.gov/telluride`.
[9] Available at `http://www.lanl.gov/telluride`.
[10] Available at `http://www.lanl.gov/telluride`.

[82] J. David Moulton, Joel E. Dendy Jr., and James M. Hyman. The black box multigrid numerical homogenization algorithm. *Journal of Computational Physics*, 142:80, 1998.

[83] Carl B. Jenssen and Per A. Weinerfelt. Parallel implicit time-accurate Navier-Stokes computations using coarse grid correction. *AIAA Journal*, 36:946–951, 1998.

[84] P. N. Brown and Y. Saad. Hybrid Krylov methods for nonlinear systems of equations. *SIAM Journal on Scientific Computing*, 11:450–481, 1990.

[85] N.N. Carlson and K. Miller. Design and application of a gradient-weighted moving finite element code I: In one dimension. *SIAM Journal on Scientific Computing*, 19:728–765, 1998.

[86] K. Miller. Nonlinear Krylov and moving nodes in the method of lines. *Journal of Computational and Applied Mathematics*, 183(2):275–287, 2005.

[87] D.R. Fokkema, G.L.G. Sleijpen, and H.A. Van Der Vorst. Accelerated inexact newton schemes for large systems of nonlinear equations. *SIAM Journal on Scientific Computing*, 19:657–674, 1998.

[88] P. Michaleris, D. A. Tortorelli, and C. A. Vidal. Tangent operators and design sensitivity formulations for transient nonlinear coupled problems with applications to elasto-plasticity. *International Journal for Numerical Methods in Engineering*, 37:2471–2499, 1994.

[89] T. L. Wilson. Stretched grid generation. Technical Report Memo N-6-88-773, Los Alamos National Laboratory, 1988.

[90] V. Maronnier, M. Picasso, and J. Rappaz. Numerical simulation of three-dimensional free surface flows. *International Journal for Numerical Methods in Fluids*, 42:697–716, 2003.

[91] X. Jiao and M.T. Heath. Common-refinement-based data transfer between non-matching meshes in multiphysics simulations. *International Journal for Numerical Methods in Engineering*, 61:2402–2427, 2004.

[92] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in Fortran*. Cambridge, 1986.

[93] P. Thevoz, J.L. Desbiolles, and M. Rappaz. Modeling of equiaxed microstructure formation in casting. *Metallurgical transactions A*, 20:311–322, 1989.

[94] M.E. Glicksman J. Lipton and W. Kurz. Equiaxed dendrite growth in alloys at small supercooling. *Metallurgical transactions A*, 18:341–345, 1987.

[95] M. Rappaz and P. Thevoz. Solute diffusion model for equiaxed dendritic growth: analytical solution. *Acta Metallurgica*, 35:2929–2933, 1987.

[96] A. Starobin, A. Kuprat, M. Stan, and S. Swaminarayan. Numerical coupling of a macro-scale casting simulation code and meso-scale equiaxed dendritic growth models. In *Proceedings of 2004 ASME Heat Transfer / Fluids Engineering Summer Conference, Charlotte, NC, USA*, July 2004.